UQAC
Université du Québec
à Chicoutimi

Évaluation de l'état de préparation des applications Android

par Guilardi Demetrio

Mémoire présenté à l'Université du Québec à Chicoutimi eu vue de l'obtention du grade de

Maître ès Sciences en Informatique

Québec, Canada

RÉSUMÉ

Le système d'exploitation Android apporte toujours de nouvelles versions et mises à jour pour améliorer la sécurité, augmenter les performances et offrir une meilleure expérience utilisateur. Lorsque Google annonce une nouvelle version, toute une chaîne de changements est déclenchée en cascade, provoquant de nombreux problèmes de compatibilité. Cette étude propose une approche pour combler le vide entre la connaissance de l'existence du phénomène de fragmentation et les métriques pour mesurer l'ampleur de ce phénomène. Notre approche commence par définir ce qu'est l'état de préparation. Après avoir défini ce qu'est la préparation, nous avons utilisé des pratiques d'analyse comparative pour définir des mesures, extraire des mesures de état de préparation et déterminer les performances de préparation. Nous avons également développé un outil nommé *AndroidPropTracker* , capable d'extraire et de mesurer la disponibilité des référentiels d'applications Android. Pour valider notre approche, nous avons effectué une analyse sur les données collectées via *AndroidPropTracker* . Les données collectées montrent que les applications Android sont devenues «moins prêtes» au fil du temps. Nos résultats ont montré que notre approche est efficace pour déterminer la préparation des applications Android. nos résultats ont montré que le degré de préparation des applications pour les nouvelles versions d'Android, ou *Readiness*, est mesurable, comparable et prévisible.

Table des matières

Table des figures

Liste des tableaux

Listings

CHAPITRE 1

INTRODUCTION

Android est un système d'exploitation OS (Open-source/logiciel libre) de Google qui est installé sur plus du trois quarts des téléphones intelligents mondialement, selon GlobalStats (2019). Android est mis à jour au moins une fois par année. Chaque nouvelle version du logiciel a pour but de résoudre certains problèmes ayant été identifiés lors des mises à jour précédentes, afin d'amener de nouvelles fonctionnalités ou encore dans l'optique de se conformer à de nouvelles lois ou règlements. Lorsqu'une nouvelle version d'Android est suggérée, elle peut être accompagnée d'une nouvelle interface de programmation d'application (API), Android Open Source Project (nd).

Les interfaces de programmation d'application (API) d'Android amènent plusieurs fonctions et procédures qui permettent aux développeurs d'accéder à différentes composantes du système d'exploitation et des dispositifs à travers un langage de programmation de haut niveau, comme Java ou Kotlin. Ces interfaces de programmation servent également de ponts connectant les capteurs de l'appareil aux applications. Un signal est envoyé aux applications en suivant des étapes bien précises : (i) Détection d'événements matériels (ii) Le développement natif d'Android (NDK) ; (iii) L'interface de programmation d'application d'Android ; (iv) Application.

Au-delà de la transmission de signaux de capteurs de bas niveau, un autre aspect important des interfaces de programmation d'application (API) est de garantir que les applications peuvent se fier sur les ressources et fonctionnalités du système. Les développeurs n'ont pas à se baser sur les spécificités du système d'Android puisque ce dernier couvre un grand nombre d'appareils différents les uns des autres, dont chacun possède des particularités qui lui sont propres.

Puisqu'une interface de programmation d'application (API) est directement liée à celle du système d'exploitation de l'application, cette interface peut amener d'importants changements, c'est-à-dire de nouvelles façons permettant aux développeurs d'accéder aux propriétés et fonctionnalités du système.

L'existence de plusieurs interfaces de programmation d'application (API) fait également partie d'un phénomène connu sous le nom de fragmentation (Han *et al.*, 2012; Wu *et al.*, 2013; Zhou *et al.*, 2014). La fragmentation peut être définie comme ayant différentes spécifications, couvrant plusieurs versions d'Android. Une fragmentation excessive a causé des problèmes au niveau de la planification, du développement et des essais des applications Android (Joorabchi *et al.*, 2013).

Pour minimiser l'effet de fragmentation, chaque version d'Android a un mode de compatibilité qui imite les comportements des versions précédentes. Cette compatibilité à rebours permet aux anciennes versions d'Android de fonctionner avec les nouvelles. Les applications sont généralement compatibles avec les nouvelles versions, bien que les développeurs aient la responsabilité de s'assurer que leurs applications vont continuer de fonctionner normalement lors de nouvelles mises à jour d'Android, ils doivent s'en assurer en testant leurs applications sur les nouvelles versions d'Android.

1.1 MOTIVATION

Les applications visant des niveaux d'interfaces de programmation d'application (API) obsolètes rendent les mises à jour de sécurité Android moins efficaces. Selon Mutchler *et al.* (2016), ces applications qui ne sont plus à jour exposent les utilisateurs à certains risques. Aussi, les différences du niveau des interfaces de programmation d'application (API) pourraient faire en sorte que certaines applications **cessent de fonctionner** lorsqu'un appareil est mis à jour avec une nouvelle version d'Android.

Selon He *et al.* (2018), 91.84% des applications d'Android n'ont pas le choix d'adresser des problèmes de compatibilité. De plus, selon Yuan *et al.* (2015), les applications qui sont compatibles avec les versions ultérieures et plus stables d'Android doivent assurer leur fonctionnement à la compatibilité.

Les potentiels problèmes connus du fait que certaines applications ne soient pas compatibles aux nouvelles versions d'Android font certainement controverse parmi des chercheurs dans le domaine. Han *et al.* (2012); Wu *et al.* (2013); Zhou *et al.* (2014) Malgré le fait qu'il y ait un manque d'analyses quantitatives et qualitatives de l'état de préparation des applications pour les nouvelles versions d'Android. Le fait de ne pas savoir l'ampleur réelle du problème créé une zone grise, dans laquelle des recherches avancées et plus profondes n'ont pas lieues. Comme dommage collatéral, il n'y a pas de pistes d'amélioration qui sont développées. Finalement, les utilisateurs finaux ont des expériences de bas niveau. Nous n'avons également pas pu trouver assez de matériel qui prouverait ou expliquerait aux développeurs l'importance d'avoir des applications prêtes à recevoir des éventuelles mises à jour d'Android.

1.2 PUBLICATIONS CONNEXES

Les publications suivantes sont étroitement liées à cette recherche :

Guilardi, Demetrio and Nicacio, Jalves and Napoleao, Bianca and Petrillo, Fabio, Android-PropTracker : Mining Lifetime Properties of An-droid Projects, in proceedings of IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems, Seoul, Republic of Korea, 2020.

Guilardi, Demetrio and Nicacio, Jalves and Napoleao, Bianca and Petrillo, Fabio, Are apps ready for new Android Releases ?, in proceedings of IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems, Seoul, Republic of Korea, 2020.

Dans les deux publications ci-dessus, nous avons, avec succès, appliqué notre approche afin de répondre aux deux questions de recherche suivantes : RQ1 : Les applications sont-elles prêtes pour les mises à jour d'Android ? RQ2 : Quels sont les délais pour s'adapter aux nouvelles versions d'Android, s'il y en a ? Mes résultats ont montré qu'avec le temps, les applications sont devenues « moins prêtes ». J'ai découvert que 76.45% des applications analysées étaient prêtes pour Android Lollipop 5.0 (API niveau 21) qui a été mise en fonction en octobre 2014. Par contre, seulement 5.46% d'entre elles étaient prêtes pour Android 10 (API niveau 29), en septembre 2019.

1.3 OBJECTIFS

L'objectif principal de cette recherche est de présenter une approche qui permet d'évaluer l'état de préparation des nouvelles versions d'Android. Notre approche comble les manques que

nous retrouvons entre connaître l'existence du phénomène de fragmentation ainsi que la manière de mesurer, avec les métriques, l'ampleur réelle du phénomène. Notre approche mesure, compare et prédit l'état de préparation des applications pour les nouvelles versions d'Android.

Nous avons également comme objectif la création d'un algorithme d'extraction de mesure permettant d'offrir une extraction et une façon de mesurer l'état de préparation de manière faisable. Un outil qui met en place un algorithme d'extraction de mesure, étant capable d'extraire et de mesurer efficacement l'état de préparation.

Nous voulons également proposer une approche étant en mesure d'aider les développeurs, les chercheurs ainsi que les utilisateurs d'Android de manière claire, efficace et fiable.

1.4 CONTRIBUTIONS

Avec cette approche, nous proposons une évaluation de l'approche de l'état de préparation dans l'optique de combler le manque qui existe entre le fait de savoir que le phénomène de fragmentation existe ainsi que la façon de mesurer l'ampleur réelle du phénomène. Notre approche définit le concept d'état de préparation, introduit la façon de mesurer ce dernier et amène une façon de déterminer la performance de cet état de préparation.

Notre recherche introduit une définition de l'état de préparation, des métriques, des mesures et de l'approche d'évaluation. Ces éléments concluent que l'état de préparation est mesurable, comparable et prévisible. En s'attardant à l'état de préparation d'une perspective d'Android, comme vu dans la section 5.2, la figure 5.1, l'évolution de l'adhésion vers une version n'est pas seulement mesurable, elle est également comparable avec d'autres versions.

Notre approche conclut également que l'état de préparation est prévisible. Lorsqu'on s'attarde à une perspective de version d'Android, il est également possible de mesurer l'état de préparation dans un temps précis.

Nous contribuons à l'aide d'un instrument qui mesure l'état de préparation nommé *Android-PropTracker* , capable de suivre les changements de l'état de préparation tout au long de la durée de vie de chaque application dans une liste donnée.

La suite de ce mémoire s'organise de la manière suivante : La section 2 présente les concepts en ce qui concerne les niveaux des interfaces de programmation d'applications (API) et les études connexes. La section 3 décrit notre approche permettant de détecter l'état de préparation des applications d'Android. La section 4 présente *AndroidPropTracker* , un outil que nous avons créé pour extraire et mesurer l'état de préparation. La section 5 décrit comment notre approche a été appliquée et testée, nous rapportons les résultats obtenus à travers les différents tests tout en évaluant les données produites et les résultats. La section 6 présente les résultats qui révèlent les implications qui affectent le champ de recherche du développement d'Android et les chercheurs du domaine, nous présentons aussi les implications qui visent les politiques de Google ainsi que les utilisateurs d'Android. Finalement, la section 7 conclut cette recherche et discute des recherches futures.

CHAPITRE 2

CONTEXTE

Android est un système opérateur complexe et sophistiqué avec plusieurs parties et composantes. Ce chapitre décrit ce qui a mené au développement de cette recherche, les connaissances antérieures nécessaires afin de réellement comprendre le cœur de ce mémoire. Dans ce chapitre, les composantes, l'historique et les informations techniques d'Android seront détaillées.

2.1 SYSTÈME D'EXPLOITATION ANDROID

[h]

Le système d'exploitation d'Android est un logiciel open-source et libre de Google, mis sur pied en 2007. Selon Alabaster (2013), le but premier d'Android était de faire fonctionner des caméras digitales, puisque le co-fondateur a affirmé, lors d'un sommet économique à Tokyo que leur système d'exploitation qui fait fonctionner les téléphones est « Exactement la même plateforme, le même système d'exploitation que nous utilisons pour fabriquer des caméras».

Android couvre plus de 75% des téléphones intelligents dans le monde, selon GlobalStats

(2019). Les mises à jour sont fréquentes, au moins une fois par année. Chaque nouvelle version a pour but de corriger certains problèmes ayant été identifiés lors des dernières mises à jour afin d'apporter de nouvelles fonctionnalités ou encore répondre à certaines lois ou règlements.

On retrouve présentement 10 versions principales d'Android sur le marché et 26 versions secondaires (Voir tableau 2.1). Le nombre de versions d'Android est l'une des facettes d'Android, Android couvre également différents appareils provenant de différents fabricants, différents écrans, diverses langues, etc. Tous ces aspects ajoutent une certaine complexité et des problèmes de cascade, mieux connu sous le nom de *Fragmentation* (Han *et al.*, 2012; Wu *et al.*, 2013; Zhou *et al.*, 2014).

Android doit faire face à la fragmentation, un phénomène causé par le système d'exploitation qui fait fonctionner les téléphones, les tablettes et autres appareils ayant différentes particularités techniques.

D'un côté plus superficiel, le problème de fragmentation pourrait être défini comme ayant un seul système d'exploitation couvrant des appareils qui ont différentes spécificités techniques comme la taille de l'écran, la mémoire et la capacité du processeur, créant des problèmes de compatibilité lorsqu'une même application est censée s'exécuter sur plusieurs appareils. Malgré le fait que ce phénomène pourrait être décrit simplement, ses racines ont de fortes connections à programmation de bas niveau et architectures informatique. Selon Kosarevsky & Latypov (2015), les avertissements pour le portage des bibliothèques natives sont l'accès à la mémoire (alignement de structure et remplissage), l'ordre des octets (endianness), les conventions d'appel et les problèmes de point flottante.

Comme d'autres systèmes d'exploitation, Android ne permet pas aux applications un accès direct aux ressources du système. Android a une interface qui permet aux applications d'employer les ressources du système, une interface de programmation d'applications (API).

2.2 LES INTERFACES DE PROGRAMMATION D'APPLICATIONS (API) D'ANDROID ET LEURS NIVEAUX

Les interfaces de programmation d'applications (API) sont des interfaces qui permettent aux applications d'utiliser les systèmes d'exploitation et leurs fonctionnalités. Les niveaux d'interfaces de programmation d'applications (API) d'Android sont des versions de développement identifiées par un numéro de version qui est progressif. Il peut arriver qu'il y ait plusieurs versions d'Android associées au même niveau d'interface de programmation d'applications (API), ce qui veut dire que le système d'exploitation (OS) a été mis à jour, mais l'interface entre ce dernier et les applications reste le même.

Le tableau 2.1 montre la liste de toutes les versions d'Android jusqu'à présent, leur nom de code, les numéros de «build» ainsi que les niveaux des interfaces de programmation d'applications (API). La dernière colonne dresse les dates de sorties de chaque version.

Les noms de code	Version	API	Date de sortie
Android 11	11	API level 30	09/08/20 (Android Developers Blog, 2020)
Android 10	10	API level 29	09/03/19 (Android Developers Blog, 2019b)
Pie	9	API level 28	08/06/18 (Android Developers Blog, 2018)
Oreo	8.1.0	API level 27	10/25/17 (Android Developers Blog, 2017a)
Oreo	8.0.0	API level 26	08/21/17 (Android Developers Blog, 2017c)
Nougat	7.1	API level 25	10/19/16 (Android Developers Blog, 2016a)
Nougat	7.0	API level 24	08/22/16 (Android Developers Blog, 2016b)
Marshmallow	6.0	API level 23	10/05/15 (Android Blog, 2015)
Lollipop	5.1	API level 22	03/09/15 (Android Developers Blog, 2015)
Lollipop	5.0	API level 21	10/20/14 (Android Developers Blog, 2014)

Les noms de code	Version	API	Date de sortie
KitKat	4.4w	API level 20	10/31/13 (Android Developers Blog, 2013)
KitKat	4.4 - 4.4.4	API level 19	10/31/13 (Android Developers Blog, 2013)
Jelly Bean	4.3.x	API level 18	07/24/13 (Android Blog, 2013)
Jelly Bean	4.2.x	API level 17	11/13/12 (Android Developers Blog, 2012b)
Jelly Bean	4.1.x	API level 16	06/27/12 (Android Developers Blog, 2012a)
Ice Cream Sandwich	4.0.3 - 4.0.4	API level 15	12/16/11 (Android Developers Blog, 2011e)
Ice Cream Sandwich	4.0.1 - 4.0.2	API level 14	10/18/11 (Android Developers Blog, 2011d)
Honeycomb	3.2.x	API level 13	07/15/11 (Android Developers Blog, 2011c)
Honeycomb	3.1	API level 12	05/10/11 (Android Developers Blog, 2011b)
Honeycomb	3.0	API level 11	02/22/11 (Android Developers Blog, 2011f)
Gingerbread	2.3.3 - 2.3.7	API level 10	02/09/11 (Android Developers Blog, 2011a)
Gingerbread	2.3 - 2.3.2	API level 9	12/06/10 (Android Developers Blog, 2010c)
Froyo	2.2.x	API level 8	05/20/10 (Android Developers Blog, 2010b)
Eclair	2.1	API level 7	01/11/10 (Android Developers Blog, 2010a)
Eclair	2.0.1	API level 6	12/03/09 (Android Developers Blog, 2009d)
Eclair	2.0	API level 5	10/27/09 (Android Developers Blog, 2009e)
Donut	1.6	API level 4	09/15/09 (Android Developers Blog, 2009c)
Cupcake	1.5	API level 3	04/27/09 (Android Developers Blog, 2009b)
(no codename)	1.1	API level 2	02/09/09 (Android Developers Blog, 2009a)
(no codename)	1.0	API level 1	09/23/08 (Android Developers Blog, 2008)

Tableau 2.1 – Les noms de code, Balises, Numéros de «build» (Android Open Source Project, nd) et les dates de sortie

La figure 2.1 illustre la chronologie des sorties d'interfaces de programmation d'applications (API) des cinq dernières années. Il y a eu sept sorties d'interfaces de programmation d'applications (API) de 2016 à 2020, les interfaces de programmation d'application 24 et 25 en 2016, 26 et 27 en 2017, 28 en 2018, 29 en 2019 et 30 en 2020.

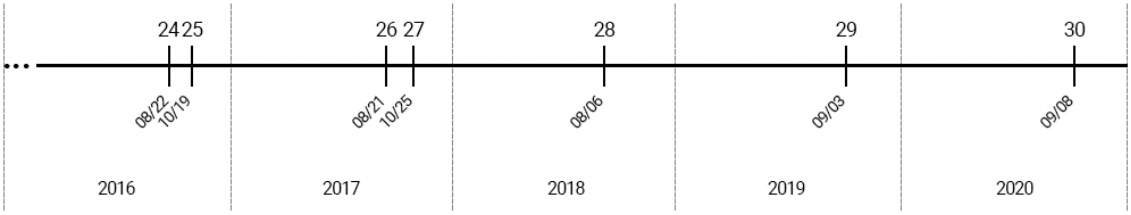

Figure 2.1 – La représentation de la chronologie des sorties d'interfaces de programmation d'applications. Les barres représentent chaque sortie des cinq dernières années, de 2016 à 2020.

Malgré le fait que la version Android Pie (niveau 28 d'API) est sortie depuis juin 2018, un peu plus de 10 % des utilisateurs l'utilisent. C'est connu que plus de 60% des utilisateurs continuent d'utiliser Android Marshmallow, Nougat et Oreo (Niveau d'API 23 et 27) (Android Developers, 2019).

Avec le temps, il y a eu plusieurs nouvelles versions d'Android, avec un grand nombre de changements au niveau du fonctionnement ainsi que différentes façons pour les développeurs d'accéder aux propriétés et fonctionnalités du système. Cela a comme effet de causer le phénomène connu sous le nom de fragmentation. Selon Wei *et al.* (2016), l'effet de fragmentation amène des défis sans précédents pour les développeurs d'applications, puisqu'il existe plusieurs versions différentes des systèmes d''exploitation d'Android sur un grand nombre d'appareils qu'on peut qualifier d'hétérogènes.

Afin de minimiser l'effet de fragmentation et atténuer le risque potentiel d'interopérabilité des applications lors d'une sortie d'Android, deux efforts doivent être faits : (Android Developers Blog, 2017b; Android Developers, ndc)

(i) La compatibilité descendante : Google fait un effort pour s'assurer que tout le monde a une expérience positive, focusant sur des améliorations constante concernant la sécurité ainsi que la performance en rendant disponible la compatibilité descendante. Cela signifie que les applications qui ont été créées pour des versions moins récentes d'Android peuvent également fonctionner sur les nouvelles versions.

(ii) La compatibilité ascendante : Les développeurs ont leur rôle à jouer pour garantir que leurs applications sont compatibles avec les nouvelles versions d'Android. Ils doivent tester leurs applications avec les interfaces de programmation d'applications qui viennent avec les nouvelles versions d'Android en réalisant un test de compatibilité ascendante.

La figure 2.2 illustre une union de compatibilité descendante et ascendante. L'appareil illustré couvre une version d'Android avec un niveau actuel d'interfaces de programmation d'applications (API) et la mise en place de la compatibilité descendante avec des niveaux plus anciens d'API. L'application illustrée fonctionnent d'un niveau d'API k et plus, elle est construite en utilisant les fonctionnalités d'Android et les comportements définis par un niveau d'API n et l'application a été testée avec le niveau actuel d'API. Les développeurs et Google garantissent que l'application fonctionnera bien sur l'appareil.

Les applications sont configurées pour fonctionner à l'intérieur d'un éventail de niveaux d'API. Une application moderne d'Android qui utilise Gradle comme outil de *build* va définir son niveau d'API sous forme de fichier *build.gradle* . Les sections suivantes décrivent le processus de *build*.

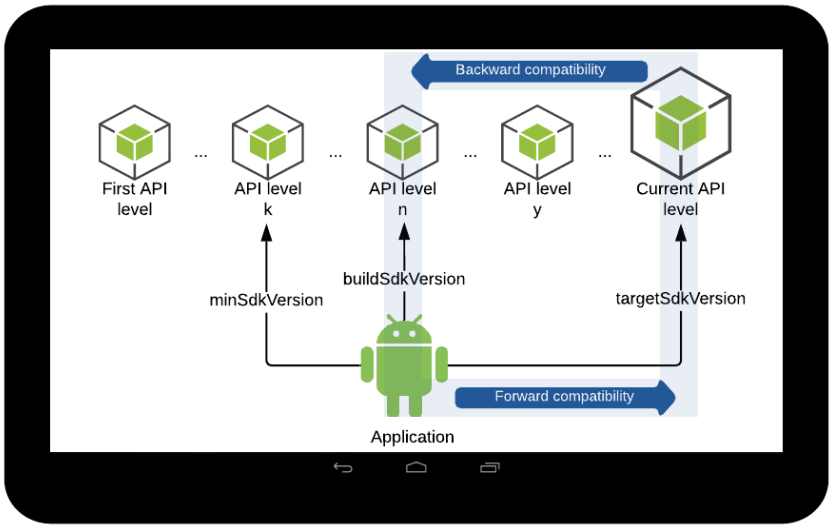

Figure 2.2 – La compatibilité ascendante et descendante jointes

2.3 LE PROCESSUS DE PRODUCTION DES APPLICATIONS ANDROID

Les projets d'Android sont écrits sous différents langages comme *Java*, *Kotlin* et *C++*. Après le codage d'une applications, il faut qu'elle soit produit pour fonctionner sur des appareils ou émulateurs. Le processus de production peut être supporté par différents outils, tels que *Maven*, *Ant* ou *Gradle*. Malgré le fait que plusieurs outils soient disponibles, Google recommande *Gradle* pour créer des applications Android. *Gradle* est également l'outil de production par défaut du Studio Android (Android Developers, ndb).

Le processus de production a deux étapes principales : la compilation et le «packaging». L'étape de compilation est responsable d'unifier le code source du logiciel, les fichiers de ressource et les bibliothèques, générant des fichiers bytecode format, les fichiers *Dalvik executable* (DEX) L'étape du «packaging» joint les fichiers DEX avec d'autres ressources, les compresse et les joint dans un fichier Android package (APK). La figure 2.3 illustre une version simplifiée du processus de fabrication d'Android.

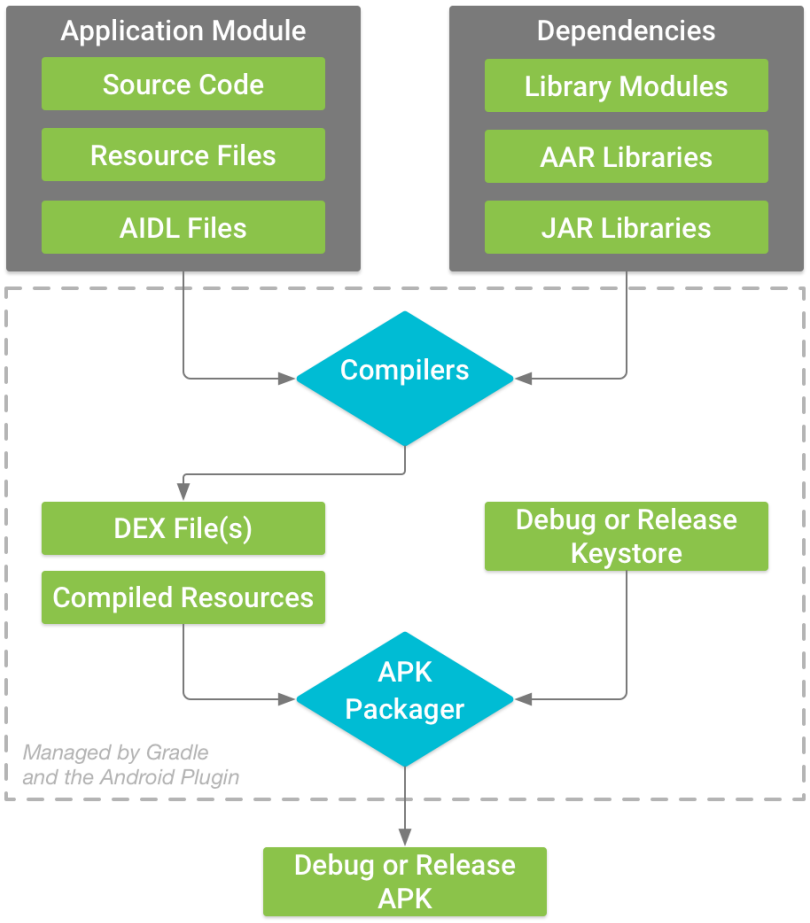

Figure 2.3 – Le processus de production d'Android (Android Developers, ndb)

2.3.1 LES PROJETS ET LES TÂCHES

Gradle est basé sur le concept de projets et tâches. *Gradle builds* sont des formules d'un ou plusieurs projets. Dans cette section, les projets sont mentionnés sous forme de *Gradle*, et non de projets d'applications Android.

Un projet peut représenter différentes artéfacts, comme des bibliothèques, des applications, des composants ou des fichiers JAR. Les projets peuvent aussi être une représentation de tâches, des choses à faire et pas forcément à construit. Un projet peut représenter, par exemple, le déploiement d'une application à un environnement (Gradle, nda).

Les tâches offrent des services aux projets. Elles représentent un travail éxécute pour la construction. Une tâche peut compiler un code source, des fichiers compactés, pack, générer de la documentation, copier des fichiers, etc (Gradle, nda).

2.3.2 LE FICHIER BUILD.GRADLE

Gradle commence en exécutant la commande `gradle`. La commande `gradle` va chercher les fichiers *build.gradle* , aussi connus sous le nom de scripts de onstruction. Le script `build` définit un projet et ses tâches (Gradle, nda).

Gradle utilise un langage spécifique du domaine (DSL) dans ses fichiers *build.gradle* , disponibles en Groovy et Kotlin. Selon Gradle (ndb), «un script fait avec *Groovy* peut contenir n'importe quel élément du langage *Groovy*. Un script fait avec *Kotlin* peut contenir n'importe quel élément du langage *Kotlin*. *Gradle* assume que chaque script est encodé en utilisant UTF-8».

```
// Local variable example
def destination = "/var/www"
task copy(type: Copy) {
    from "source"
    into destination
}

// Extra property example
ext {
    version = "1.0.0.RELEASE"
}
task printProperties {
    doLast {
        println version
    }
}
```

Listing 2.1 – Déclaration de variables locales et de propriétés supplémentaires dans un fichier *build .gradle*

Les variables locales et les propriétés supplémentaires

Un *build.gradle* contient seulement deux types de variables, les variables locales et les propriétés supplémentaires. Les variables locales sont seulement visibles dans le cadre dans lequel elles ont été déclarées. D'un autre côté, les propriétés supplémentaires sont fixées et attachées à un objet et peuvent être lues n'importe où via l'objet en question.

La listage 2.1 exemplifie la déclaration et l'utilisation d'une variable locale et d'une propriété supplémentaire. La *destination* est la variable locale et la *version* est une propriété supplémentaire.

2.3.3 COMPILATION AVEC LA GAMME DES VERSIONS D'ANDROID

Gradle, Play Store, Android OS et d'autres applications sont quatre variables pour extraire le mode de compatibilité des applications avec les différentes versions d'Android sur lesquelles elles

ont été créées pour fonctionner (Android Developers, ndd,n,n).

— `minSdkVersion`. Définit le niveau minimum d'API qu'une application supporte pour fonctionner. Le système d'Android ne permettra pas à l'application d'être installée si le niveau d'API en marche est plus bas que la valeur spécifié dans cet attribut (Android Developers, ndc).

— `compileSdkVersion`. Définit le niveau d'API Android sur lequel l'application est compilée. L'application utilise les fonctionnalités d'Android incluses dans le niveau spécifique d'API ou un niveau plus bas (Android Developers, ndb).

— `targetSdkVersion`. Définit le niveau d'API d'Android sur lequel l'application est destinée à fonctionner et testée. Ça informe le système que le développeur a testé contre la version visée et qu'il ne devrait permettre aucune compatibilité pour maintenir la compatibilité ascendante avec la version visée (Android Developers, ndc).

— `maxSdkVersion`. Définit le niveau maximum d'API qu'une application peut supporter. Le système d'Android ne permettra pas à l'application d'être installée si le niveau d'API actuel est plus haut que la valeur spécifique à cet attribut (Android Developers, ndc).

minSdkVersion et *targetSdkVersion* sont définies dans *build.gradle* ou dans les fichiers *AndroidManifest.xml* . *compileSdkVersion* est définie dans les fichiers *build.gradle* seulement. *maxSdkVersion* est définie dans les fichiers *AndroidManifest.xml* seulement.

La gamme de versions visées dans le fichier *AndroidManifest.xml*

Selon Android Developers (2020), chaque projet d'application doit avoir un fichier *AndroidManifest.xml* . Un *AndroidManifest.xml* décrit des informations essentielles à propos des applications. À travers ces informations essentielles, on retrouve des attributs de *minSdkVersion, targetSdkVersion*

```
<manifest>
    <uses-sdk android:minSdkVersion="21"
              android:targetSdkVersion="29"
              android:maxSdkVersion="30" />
</manifest>
```

Listing 2.2 – La gamme de versions visées dans le fichier *AndroidManifest.xml*

et maxSdkVersion. La *compileSdkVersion* ne peut être définie sur manifest, voir la section suivante pour plus de détails.

Les attributs de *minSdkVersion, targetSdkVersion and maxSdkVersion* sont fixées de manière spécifique dans les fichiers *AndroidManifest.xml* . La listage 2.2 présente un exemple de comment ces attributs sont fixés dans les fichiers *AndroidManifest.xml* .

La gamme de versions visées dans le fichier *build.gradle*

Google recommande aux applications Android d'être réunies avec *gradle*, d'informer le compilateur à travers *minSdkVersion, compileSdkVersion and targetSdkVersion*. *targetSdkVersion* détermine avec quelles versions elles doivent fonctionner. Les versions qui ont la capabilité et les fonctionnalités que les applications ont. Les applications doivent également informer les compilateurs de deux autres variables pour constituer un éventail complet supporté par les versions d'Android. *minSdkVersion* configure gradle avec un niveau minimum d'API (la version d'Android) que les applications supportent (Android Developers, ndb). *targetSdkVersion* configure gradle avec le niveau maximum d'API (la version d'Android) sur lequel les applications ont été destinées à fonctionner (Android Developers, ndc). L'éventail des niveaux d'API est défini par trois variables dans les fichiers *build.gradle* sur les projets d'Android qui utilisent *Gradle*.

Une propriété *gradle* est définie de différentes manières. La listage 2.3 présente un exemple

```
android {
    compileSdkVersion 29
    buildToolsVersion '28.0.3'
    defaultConfig {
        minSdkVersion 21
        targetSdkVersion 29
    }
}
```

Listing 2.3 – Déclaration des propriétés de la plage de versions dans le fichier *build.gradle*

de comment ces attributs sont fixés dans les fichiers *build.gradle* :

2.4 TRAVAIL CONNEXE

2.4.1 PROBLÈMES DE FRAGMENTATION DES API

Détecter et comprendre les problèmes de fragmentation. Wei *et al.* (2018) a mené une étude empirique sur 220 problèmes de compatibilité provenant de cinq sources ouvertes d'applications d'Android. Ils ont procédé à différentes entrevues et sondages afin d'avoir une perception sur la façon dont les praticiens gèrent les problèmes de fragmentation. Ils ont également catégorisé les problèmes de fragmentation, les problèmes de divergence qui affecteraient un appareil précis, les fonctionnalités des applications ou encore l'expérience de l'utilisateur. De plus, ils ont proposé un outil pour détecter les problèmes de compatibilité automatiquement et directement sur les applications Android : FicFinder.

Les problèmes de compatibilité d'API d'Android. Huang *et al.* (2018) ont analysé 100 réels retours de problèmes de compatibilité sur 20 applications open-source d'Android sur GitHub. Ils ont également développé un outil statistique.

La péremption de l'API d'Android. Our study consider the whole projects lifetime and all their forward compatibility changes. Mutchler *et al.* (2016) a analysé 1.2 millions d'applications et ont conclu que 93% cible des versions de plate-forme obsolètes et ont une moyenne de l'obsolescence de 686 jours. Leur méthode d'étude était d'analyser ces applications en téléchargeant leurs fichiers binaires, en faisant la décompilation et en récupérant les valeurs de compatibilité ascendante. Leur étude considère l'entièreté de la vie du projet et ses changements au niveau de la compatibilité ascendante.

Impacts dans les applications causés par les mises à jour d'Android OS. Yang *et al.* (2018) ont fait une séries de scripts qui utilisent le code source des applications pour trouver et signaler les classes définies par l'utilisateur qui pourraient être affectées si les applications mettent à jour la version de leur plateforme SO. Ces scripts extraient le nombre et le pourcentage de classes d'utilisateur qui seraient affectées dans une éventuelle mise à jour de niveau d'API (changement de *targetSdkVersion*).

Caractérisation des niveaux d'API obsolètes. Li *et al.* (2020) ont réalisé une extraction systématique du code source du cadre d'Android et analysé 10 000 applications réelles d'Android. Ils ont développé un outil appelé CDA pour extraire les codes sources d'Android, repérer les fonctions et classes désuètes. Ils ont ensuite croisé ces résultats avec les 10 000 applications sélectionnées, identifiant le nombre de fois les fonctions et classes désuètes ont été utilisées par l'application, par niveau d'API.

2.4.2 L'EXPLORATION ET L'ANALYSE DU APP STORE

L'exploration et l'analyse du App Store. Harman *et al.* (2012) a comparé l'extraction du App Store à l'extraction de références. Ils considèrent les applications mobiles publiées en magasin

par les métriques métadonnées puisque les codes sources ne sont pas disponible au téléchargement. Ils ont utilisé des techniques d'explorations de données afin d'extraire de l'information, puis ils ont combiné les données extraites avec des informations facilement accessibles pour accéder aux aspects techniques de l'application, aux clients et aux aspects commerciaux.

Historique des *commits* des réelles applications d'Android. Geiger *et al.* (2018) ont développé un outil de dépôt d'extraction nommé AndroidTimeMachine. Leur outil est capable de joindre les applications Andoid publiées sur Play store avec leur code GitHub. Ils combinent l'information provenant de GitHub et du Play Store afin de créer une base de données incluant les métadonnées de GitHub et du Play Store ainsi qu'une historique simple. Leur outil est un projet open-source disponible sur GitHub et leur base de données est également disponible au public.

2.4.3 LIMITES DES APPROCHES EXISTANTES

Cette sous-section énumère les limitations des travaux connexes que nous avons trouvées.

(i) Wei *et al.* (2018). Leur travail a pour but de trouver et catégoriser les problèmes de fragmentation, tandis que notre travail vise à quantifier les applications qui pourraient potentiellement avoir des problèmes de fragmentation avec le temps.

(ii) Huang *et al.* (2018). Leur étude analyse seulement 20 applications, cherchant les archives pour des solutions spécifique pour régler les comportements des API.

(iii) Mutchler *et al.* (2016). Leur méthodologie consistait à télécharger et à décompiler les binaires de l'application. Leur étude ne tient pas non plus compte des changements au fil du temps.

(iv) Yang *et al.* (2018). Leur travail détecte les classes qui seraient potentiellement affectées par une mise à jour Android. Ils ont créé des scripts pour analyser les codes sources et générer des

rapports.

(v) Li *et al.* (2020). Cette étude analyse l'utilisation de méthodes d'API désuètes, tandis que la nôtre analyse l'utilisation des niveaux d'API en soi.

(vi) Harman *et al.* (2012). Ils ont analysé 32 108 applications payées de Blackberry. Leurs résultats corrèlent le prix, les avis des utilisateurs, le rang des applications ainsi que les téléchargements.

(vii) Geiger *et al.* (2018). Leur étude ne tient pas compte des changements au fil du temps.

CHAPITRE 3

APPROCHE

Dans ce chapitre est présentée notre approche permettant d'évaluer l'état de préparation des application. Premièrement, nous nous devons de définir ce qu'est l'état de préparation. Après avoir défini cette dernière, nous utilisons des pratiques «benchmarking» afin de définir les métriques, d'extraire l'état de préparation ainsi que mesurer et déterminer la performance de l'état de préparation.

Selon Oxford Learner's Dictionaries Online (2020), le «benchmark» est «quelque chose qui peut être mesurée et utilisée comme standard afin d'être comparé à d'autres choses». Le «benchmarking» est un sujet fortement discuté, Anand & Kodali (2008) ont réalisé une révision de littérature du «benchmarking» et ont conclu que les définitions du terme varient, c'est-à-dire qu'on peut trouver 49 définitions de «benchmarking». Toujours selon Anand & Kodali (2008), dans certains cas, un modèle de «benchmark» a été utilisé de façon unique pour performer un type particulier de «benchmarking».

Dans notre approche, le «benchmarking» placera les applications dans un ensemble de données comparable, en comparant l'indicateur de l'état de préparation (*targetSdkVersion*) au fil du temps. Afin de déterminer si les applications étaient et sont prêtes ainsi que leur éventuel état de

préparation, notre approche du «benchmarking» emploi des mesures d'intervalles et des échelles afin que la différence entre les valeurs soit significative (Briand *et al.*, 1996).

3.1 LA DÉFINITION DE L'ÉTAT DE PRÉPARATION

Selon Merriam-Webster's Collegiate Dictionary Online (1999), l'état de préparation est « la qualité ou l'état d'être prêt (pour quelque chose)». Comme mentionné dans la section 2.3.3, le «gradle» d'Android, le Play Store et le OS dépendent de quatre attributs afin d'extraire la compatibilité d'une application. Parmi les quatre attributs, *minSdkVersion* et *maxSdkversion* agissent comme des filtres. Lorsque les applications utilisent ces deux attributs, elles informent l'écosystème d'Android qu'elles ne vont pas fonctionner avec les versions d'Android moindres ou égales à *minSdkVersion* ou meilleures ou égales à *maxSdkVersion*. Au contraire, lorsque les applications utilisent les attributs de *compileSdkVersion* et *targetSdkVersion*, elle informent spécifiquement l'écosystème d'Android qu'elles sont destinées à fonctionner avec les versions visées.

Afin de vérifier avec quelle version la plus récente les applications sont supposées fonctionner, l'attribut de *targetSdkVersion* est observé (voir section 2.3.3). *targetSdkVersion* informe le système que le développeur a testé contre la version visée et que le système ne devrait pas permettre de compatibility behaviours (Android Developers, ndc). La *targetSdkVersion* est établie dans des fichiers *AndroidManifest.xml* et *build.gradle* , malgré que selon Android Developers (2020, ndc), les applications qui ont été conçues avec Android Studio doivent définir *targetSdkVersion* dans les fichiers *build.gradle* et la valeur définie dans les fichiers *AndroidManifest.xml* est ignorée.

Définition de l'état de préparation
Avoir *targetSdkVersion* établie à un niveau spécifique d'API, dans un temps donné.

Après avoir défini ce qu'est l'état de préparation, le processus que nous proposons est composé de trois étapes. Étape 1 : Définir les métriques de l'état de préparation ; Étape 2 : Extraire les mesures de l'état de préparation ; Étape 3 : Déterminer la performance de l'état de préparation.

3.2 ÉTAPE 1 : DÉFINIR LES MÉTRIQUES DE L'ÉTAT DE PRÉPARATION

Dans notre approche, nous proposons un processus de métrique unique qui est simple, mais bien défini : l'état de préparation dans le temps.

Une liste des sorties d'Android est composée afin de créer une échelle d'intervalles (Briand *et al.*, 1996), avec laquelle une application sera comparée. Cette échelle déterminera si une application était prête pour une nouvelle version d'Android au moment de la sortie de celle-ci.

Les nouvelles versions d'Android sont annoncées publiquement par Google sur le site web des développeurs d'Android, voir tableau 2.1. Puisque qu'Android OS est utilisé sur plus des trois quarts des téléphones intelligents mondialement (GlobalStats, 2019), il est difficile de déterminer le moment exact où une nouvelle version commence à être distribuée aux utilisateurs (Wei *et al.*, 2016; Mutchler *et al.*, 2016). Malgré le fait que la distribution soit difficile à retracer, une nouvelle version d'Android n'est pas distribuée aux utilisateurs avant une annonce formelle de la part de Google.

3.3 ÉTAPE 2 : EXTRAIRE LES MESURES DE L'ÉTAT DE PRÉPARATION

Ayant déterminé que la valeur de *targetSdkVersion* déclare l'ultime version d'Android avec laquelle les applications sont destinées à fonctionner, nous devons également considérer l'aspect du temps. Les applications peuvent changer leur valeur de *targetSdkVersion* au fil du temps, ce qui veut

dire que la façon dont elles se déclarent prêtes à fonctionner avec différentes versions d'Android peut varier.

Ayant également déterminé les métriques de l'état de préparation, j'introduis, dans cette étape, un algorithme permettant de détecter les changements au niveau de l'état de préparation ainsi qu'une méthode qui permet d'extraire des mesures.

3.3.1 ALGORITHME DE DÉTECTION DES CHANGEMENTS D'ÉTAT DE PRÉPARATION

Dans notre approche, dans le but de détecter des changements, une liste des valeurs de *targetSdkVersion* est nécessaire, en dates d'ordre décroissant. Les items de cette liste sont définis comme une structure des données par paires, où la première tient la valeur de date et la deuxième tient les valeurs de *targetSdkVersion*.

Dans le but de détecter les changement de valeur de *targetSdkVersion*, nous avons développé l'algorithme 1. Il filtre un tableau de paires classées en valeur selon les dates, retournant ces dernières qui contiennent des changements de valeur de *targetSdkVersion*.

L'algorithme 1 est une variante de recherche dichotomique. Il procède à une recherche dichotomique des paires de dates et valeurs, bien qu'à la place des événements d'une valeur cherchée, il cherche des changements, retournant une liste d'événements.

L'algorithme 1 peut également être utilisé pour détecter des changements dans une autre variable ou un paramètre. Par contre, pour respecter le but de rester dans le périmètre de l'état de préparation, aucune autre utilisation ne sera discutée dans cette étude. L'utilisation ultérieure est mentionnée dans la section 7.

Algorithm 1: detectPropertyChangeHistory

Input: dateValuePairs : tableau ordonné de paires date-valeur, où les premiers sont les dates et les secondes sont les valeurs targetSdkVersion ; left : indice du tableau des premières paires date-valeur (plus anciennes) ; right : indice du tableau des dernières paires date-valeur (les plus récentes) ; currentValue : dernière valeur targetSdkVersion (la plus récente) ; resultPairs : tableau initialement vide de paires, utilisé en récursivité.

Output: resultPairs : tableau de paires date-valeur contenant uniquement les modifications de targetSdkVersion

```
1  middle = left + ceil( (right - left) / 2 )
2  middleValue = dateValuePairs[middle].second
3  if middle = right then
4      if middleValue = currentValue then
5          resultPairs = push(results, dateValuePairs[middle])
6          return detectPropertyChangeHistory(dateValuePairs, left + 1,
             size(dateValuePairs), middleValue, resultsPairSet)
7      else
8          return resultPairs
9  if middleValue = currentValue then
10     left = middle
11 else
12     right = middle
13 return detectPropertyChangeHistory(dateValuePairs, left, right, currentValue,
     resultPairs)
```

La figure 3.1 illustre la chronologie des changements de targetSdkVersion. L'algorithme 1 détecte les quatre changements, des niveau d'API de 23 à 25, de 25 à 26, de 26 à 28 et de 28 à 29, retournant un tableau des paires qui contiennent les quatre changements avec leurs dates.

La listage 3.1 met en exemple la sortie de l'algorithme, comme l'exemple donné sur la figure 3.1. Chaque paire de dates et valeurs représente un changement détecté par *targetSdkVersion*, où les premières représentent les dates où les changements sont arrivés et les deuxièmes représentent à quelles valeurs la valeur *targetSdkVersion* a été modifiée.

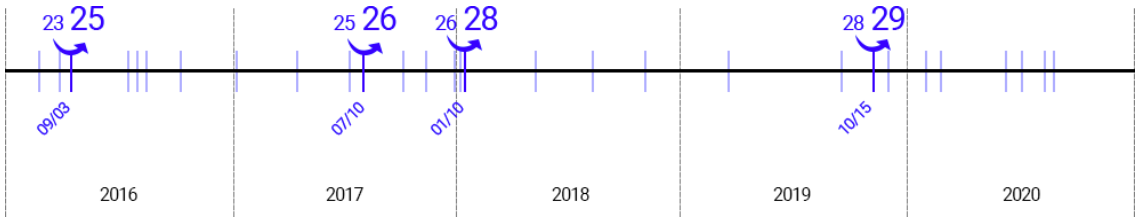

Figure 3.1 – La représentation chronologique de l'état de préparation change la détection de l'algorithme. Les lignes bleues représentent l'entrée des paires de dates et valeurs et les lignes bleu pâle représentent les paires qui n'ont pas de changement de targetSdkVersion, les lignes bleu foncé représentent les paires qui ont des changements de targetSdkVersion.

```
Array[
  Pair<Date("09/03/2016"), 25>,
  Pair<Date("07/10/2017"), 26>,
  Pair<Date("10/01/2018"), 28>,
  Pair<Date("10/15/2019"), 29>
]
```

Listing 3.1 – L'état de préparation change l'échantillon de l'algorithme de détection

L'exécution de l'algorithme 1 : Saisie d'un tableau des paires de dates et valeurs en ordre décroissant, où les dates sont les premières et les deuxièmes les valeurs de targetSdkVersion, avec des indices minimum et maximum et la la valeur actuelle (la dernière) de *targetSdkVersion* ; Calculer l'indice du milieu situé entre le minimum et le maximum ; Si l'indice du milieu est égale à celui de droite, il y a collision, Sii la valeur du milieu égale la valeur actuelle de targetSdkVersion, il y a un changement de valeur de *targetSdkVersion*, Pousser le changement détecté vers le tableau de paires, augmenter l'indice gauche, l'indice droit prend la taille des paires de dates et valeurs, et retourne l'appel récurrent de «*detectPropertyChangeHistory*», sinon, retourne le tableau des résultats des paires ; Si la valeur du milieu est égale à la valeur actuelle de *targetSdkVersion*, l'indice droit prend la valeur de l'indice du milieu, sinon, l'indice gauche prendre la valeur de l'indice du milieu ; Retourne l'appel récurrent de «*detectPropertyChangeHistory*».

3.3.2 LA MESURE : L'ÉTAT DE PRÉPARATION AU FIL DU TEMPS

Dans le but de définir si une application était prête pour une nouvelle version d'Android au moment de sa sortie, les changement de *targetSdkVersion* détectée par l'algorithme 1 est chevauché sur la liste des sorties d'Android. Ensuite, une échelle est créée, dans laquelle les applications sont comparées l'une à l'autre.

Les différences entre le moment où les applications étaient prêtes à travailler avec des versions d'Android et le moment où ces versions étaient publiées peuvent être positives ou négatives. Les valeurs positives sont celles dans lesquelles les applications sont prêtes pour des versions avant même leur sortie. Les valeurs négatives sont celles dans lesquelles les applications sont prêtes pour de nouvelles versions d'Android après leur sortie.

La figure 3.2 illustre une chronologie avec les changements de targetSdkVersion (voir figure 3.1) concernant les sorties des API (voir figure 2.1) pour les cinq dernières années. Les lignes bleues et les textes indiquent les changements d'API détectés par l'algorithme 1, des niveaux d'API 23 à 25, de 25 à 26. De 26 à 28 et de 28 à 29, retournant un tableau de paires contenant les quatre changements détectées avec leurs dates. Les lignes noires et les textes indiquent les sept sorties d'API de 2016 à 2020, les API 24 et 25 en 2016, les API 26 et 27 en 2017, l'API 28 en 2018, l'API 29 en 2019 et l'API 30 en 2020. Les épaisses lignes vertes indiquent des valeurs positives, quand l'application analysée étaient d'avance, prête pour les sorties des nouvelles versions d'Android. Les épaisses lignes rouges indiquent les valeurs négatives, quand l'application analysée était derrière, pas prête pour les sorties des nouvelles versions d'Android.

Une échelle nominale classerait si les applications étaient prêtes pour les sorties d'Android comme une classification dichotomique : oui ou non. Pour créer une échelle d'intervalle signifiante

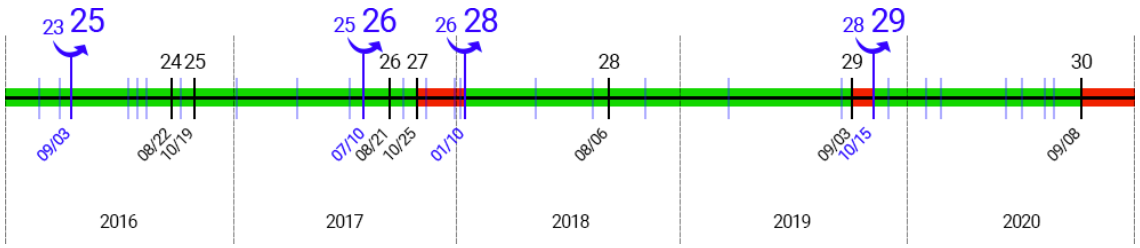

Figure 3.2 – La représentation chronologique de l'état de préparation change l'algorithme de détection (le contenu bleu, voir 3.1),des sorties d'API (le contenu noir, voir 2.1). Les épaisses lignes rouges indiquent quand l'application analysée n'était pas prête pour les versions d'Android. Les épaisses lignes vertes indiquent quand l'application analysée était prête pour les versions d'Android.

(Briand *et al.*, 1996), nous proposons que la mesure de l'état de préparation avec le temps soit considérée en tant que durée. Mesurant le délais et l'avance, à la place de oui ou non.

Pour illustrer une mesure, le tableau 3.1 contient des valeurs extraites de la figure 3.2. La première colonne contient les niveaux d'API. La deuxième colonne contient l'état prêt, lorsque l'application a été adaptée et est devenue prête pour le niveau d'API de la rangée. La troisième colonne contient les dates de sorties des niveaux d'API. La quatrième et la cinquième colonne contiennent la mesure, la différentes entre deux dates, dans deux précisions, en jours et en mois. Les valeurs positives indiquent que l'application était prête avant la sortie d'Android. Les valeurs négatives indiquent que l'application était prête après la sortie d'Android.

API level	Date prêt	Date de sortie d'Android	Différence en jours	Différence en mois
30	03/09/16	08/22/16	166	5
29	03/09/17	10/19/16	224	7
28	07/10/17	08/21/17	42	1
27	01/10/17	10/25/17	-77	-2
26	01/10/18	08/06/18	208	6
25	10/15/19	09/03/19	-42	-1
24	12/31/20	09/08/20	-114	-3

Tableau 3.1 – L'état de préparation au fil du temps. Les différences en jours et en mois entre les sorties d'Android et l'état de préparation.

La figure 3.3 contient une charte qui représente l'état de préparation au fil du temps du tableau 3.1. L'axe X représente les niveaux d'API. L'axe Y représente la durée en jours.

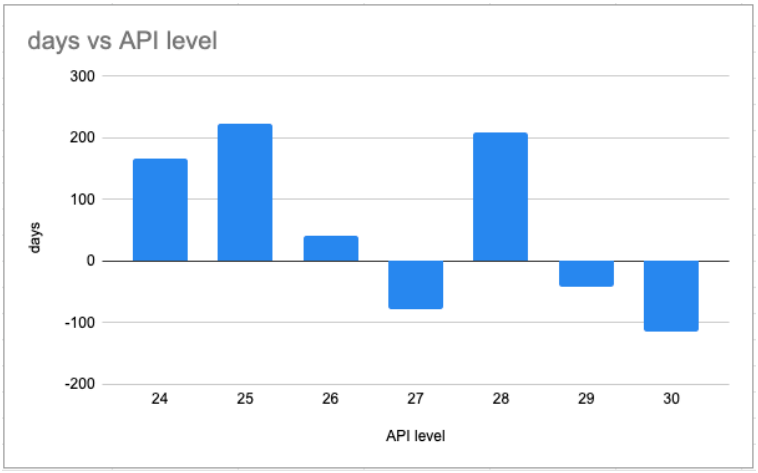

Figure 3.3 – L'état de préparation avec le temps du tableayu 3.1. Les différences en jours et en mois entre les sorties d'Android et l'état de préparation.

3.4 ÉTAPE 3 : DÉTERMINER LA PERFORMANCE DE L'ÉTAT DE PRÉPARATION

Ayant défini les métriques et mesures de l'état de préparation, cette étape consiste à introduire un moyen de déterminer la performance de l'état de préparation. Après avoir collecté les données historiques de *targetSdkVersion*, une technique statistique de régression est appliquée sur les données collectées. SelonMontgomery *et al.* (2012), la régression est une technique largement utilisée, dont l'utilité résulte du processus conceptuellement logique et d'un ensemble de variables prédictives associées. Toujours selon Montgomery *et al.* (2012), les régressions linéaires sont utilisés à différentes fins, incluant la descriptions de données, l'estimation du périmètre, la prédiction et l'estimation. Dans notre approche, une régression linéaire simple est appliquée sur les données (Montgomery *et al.*, 2012), afin de déterminer la performance de l'état de préparation.

Une régression linéaire ($y = x\beta + \varepsilon$) est appliquée sur les mesures de l'état de préparation pour déterminer la performance. y est la variable de réponse, la variable expliquée ; β est un vecteur de paramètre dimensionnel, le coefficient de régression ; ε est un vecteur de valeurs,le terme d'erreur. x est un indice analysé d'une version d'Android, le régresseur.

La figure 3.4 contient une régression linéaire appliquée sur l'état de préparation mesurée du tableau 3.1. L'axe X représente les niveaux d'API. L'axe Y représente la durée en jours. La ligne droite rouge est une régression linéaire appliquée sur les données. Pour l'exemple illustré, la performance de l'état de préparation est $-43.1 * x + 187$, où x sont des indices de versions d'Android. Les indices des versions d'Android varient de 0 à 6, où 0 indique un niveau d'API de 24 et 6 un niveau d'API de 30.

La performance de l'état de préparation de la figure 3.4 ($-43.1 * x + 187$) peut être réajustée pendant que l'application est adaptée et prête pour une nouvelle version d'Android. Aussi, appliquer

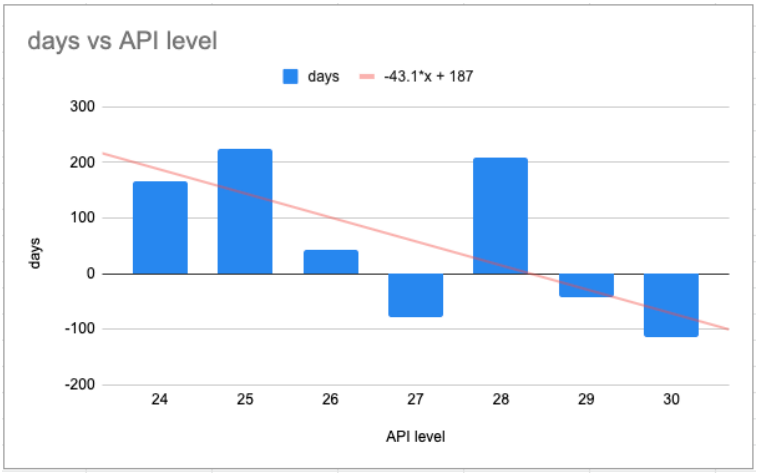

Figure 3.4 – L'état de préparation avec le temps du tableau 3.1 avec régression linéaire. Les différences en jours entre les sorties d'Android et l'état de préparation. La ligne droite rouge est une régression linéaire appliquée sur les données.

la formule $-43.1 * x + 187$, pour l'indice 7, niveau d'API 31, il est prévisible que l'état de préparation sera $-43.1 * 7 + 187$, ou -114 jours derrière.

Le coefficient de régression (β), détermine si les applications analysées ont tendance à être à jour ou pas. Des coefficients de régression positifs affirment mathématiquement que l'état de préparation est de plus en plus en avance par rapport aux versions d'Android, tandis que des coefficients négatifs affirment que l'état de préparation est de plus en plus en retard par rapport aux versions d'Android.

La performance de l'état de préparation
Le coefficient de régression détermine la performance de l'état de préparation. Les coefficients positifs indiquent que les applications ont tendance à être prêtes avant les versions d'Android. Les coefficients négatifs indiquent que les applications ont tendance à être prêtes après les versions Android.

Maintenant que nous avons défini les trois étapes principales de cette approche, nous allons introduire un outil de support.

CHAPITRE 4

OUTIL DE MESURE DE L'ÉTAT DE PRÉPARATION

Pour extraire la mesure de l'état de préparation, nous avons développé un outil d'indexation de dépôt de logiciel (Software Repository Mining Tool) appelé *AndroidPropTracker* [1]. Notre outil met en oeuvre l'algorithme de détection des changements de l'état de préparation (voir algorithme 1 chapitre 3 dans la section 3), ainsi qu'exécuter son propre algorithme d'optimisation (voir algorithme refalgo :composeRelevantCommitsList).

AndroidPropTracker est un *web crawler* de Github, qui traque les changements de *targetSdkVersion* et qui met en index tous les changements de la durée de vie de chaque dépôt dans une liste. L'outil a une liste des dépôts de GitHub comme entrée et une liste consolidée de tous les changements de valeurs de *targetSdkVersion* au fil du temps, pour chaque module de chaque projet dans la liste d'entrée, comme sortie. *AndroidPropTracker* utilise le site web de GitHub pour accéder à de l'information des projets d'Android comme des modules, commit, dates de chaque commit et la valeur de propriété de and targetSdkVersion sur chaque commit.

AndroidPropTracker suit une propriété qui obéit à l'hypothèse d'une seul dépôt, ayant la

1. Source code available at https ://github.com/deguilardi/androidPropTracker

même propriété déclarée à plusieurs reprises. Un dépôt unique de GitHub peut contenir plusieurs projets, un seul projet peut également contenir plus d'un module. De plus, chaque module classe ses propres fichiers *build.gradle* , où la valeur de la propriété tracée est fixée.

Afin d'extraire l'information sur la valeur de propriété tracée de chaque projet, il est nécessaire de lire chaque dépôt et vérifier la valeur de la propriété tracée dans chaque module de chaque projet. Qui plus est, comme je suis intéressée dans l'évolution d'une propriété au fil du temps, il est également nécessaire de tracer tous les commit faits dans chaque dépôt afin de savoir si et où la valeur de propriété a changé et quelles étaient les valeurs lui étant assignées au fil du temps.

4.1 ARCHITECTURE

AndroidPropTracker est une application orienté objet, principalement écrite en PHP 7. Les composantes du côté du client (l'interface de l'utilisateur) ont été écrites en HTML, CSS et JavaScript. De plus, le côté client utilise la bibliothèque jQuery pour des appels asynchrones et la bibliothèque chart.js pour dessiner les graphiques de sortie.

La figure 4.1 montre la vue d'ensemble de l'architecture. Le bonhomme bleu démontre un utilisateur de notre outil, qui donne une liste de dépôts à récupérer comme principale entrée (les autres champs entrées sont décrits dans la Section 4.5). Ensuite, le navigateur envoie au moteur étant du côté du serveur, qui filtre l'entrée (Section 4.2). Le moteur explore, de façon asynchrone, le dépôts, vérifie l'existence d'une cache sérialisée, récupère de GitHub si la cache n'existe pas ou lit directement de la cache locale si elle existe (Section 4.3). Lorsque chaque dépôt est récupéré, le moteur traite les données (Section 4.4), créant les sorties HTML et JS. Au moins, le navigateur de l'utilisateur restitue l'ensemble des données et des graphiques.

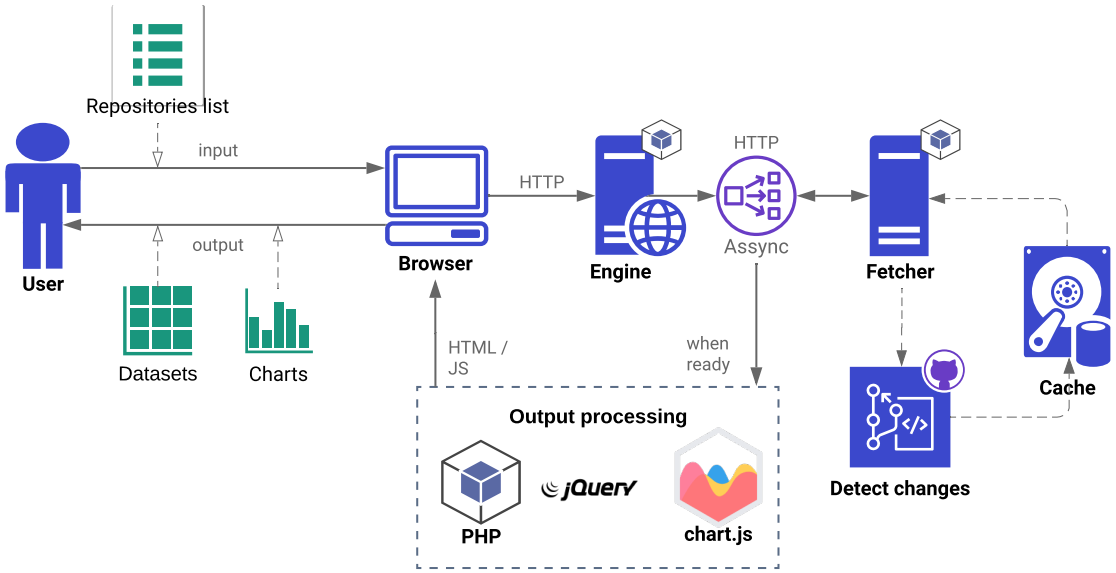

Figure 4.1 – Vue d'ensemble de l'architecture

4.2 INPUT FILTERING

Le moteur filtre chaque entrée d'utilisateur avant de récupérer les données de GitHub. Premièrement, le moteur vérifie la liste de dépôts donnée par l'utilisateur et supprime les dupliqués. Ensuite, le moteur retire de la liste les dépôts qui contiennent des chaînes prédéterminées dans leur url : *react*, *xamarin*, *cordova*, *learning*, *test*, *sample*, *udacity*. Les termes filtrés peuvent être personnalisés en modifiant le fichier *config.inc* pour changer, ajouter ou effacer les termes.

AndroidPropTracker a un fichier de configuration qui contient une variable utilisée pour filtrer les dépôts par nom, comme il est requis. La listage 4.1 contient un exemple de comment *config.inc* est utilisé pour filtrer les dépôts. Dans l'exemple, tous les dépôts qui contiennent n'importe quels termes sous forme de *$ignoredRepositoriesNames* dans leurs noms sont filtrés.

```
$ignoredRepositoriesNames = array(
    "react", "xamarin", "cordova", "learning",
    "test", "sample", "udacity"
);
```

Listing 4.1 – Exemple de filtre de dépôts de config.inc. Tous les dépôts qui contiennent n'importe quel terme sous forme de *$ignoredRepositoriesNames* dans leurs noms sont filtrés.

4.3 INDEXATION DES DONNÉES

Après que les dupliqués aient été supprimés et les url filtrés, les données commencent à être explorées. Dans cette phase, la première étape est de vérifier les modèles pour assurer que les dossiers requis sont en place et que le dépôt possède la bonne structure lui permettant d'être analysé. Deuxièmement, l'outil recherche pour des projets d'Android (un dépôt peut contenir plus qu'un projet). Dans cette étape, les fichiers *build.gradle* sont recherchés pour s'assurer que seulement les projets avec gradle sont récupérés.

Une fois que les projets restants de la liste sont tous vérifiés comme étant des projets Android avec gradle, ces projets sont récupérés de manière asynchrone par l'extracteur un par un. *AndroidPropTracker* récupère tous les changements de la propriété suivie tout au long de chaque projet. De plus, une barre de progrès est montrée avec le progrès de cette phase.

La figure 4.2 contient un organigramme détaillant les composantes internes de l'étape Détection des changements (Voir figure 4.1). Le flux débute en considérant l'état actuel du projet qui est suivi. Dans d'autres mots, le tout dernier commit de la branche principale. *AndroidPropTracker* détecte tout d'abord *(1)* les modèles du projet et crée ensuite une liste de tous les fichiers qui doivent être analysés, tels les fichiers *build.gradle* . Lors de la deuxième étape *(2)*, l'outil trouve et répertorie les fichiers externes connexes, ceux inclus par gradle à travers l'instruction informatique apply from, c'est nécessaire parce que parfois la valeur de la propriété suivie est fixée dans un de ces fichiers

(pour plus de détails, voir algorithme 2). La troisième étape *(3)* consiste à charger tous les commits de chaque fichier dans les les listes précédentes pour ensuite fusionner ces commits en gardant une seule ligne temporelle qui est linéaire parmi toutes celles-ci. La quatrième *(4)* étape est d'analyser le contenu de tous les fichiers et rechercher la valeur de la propriété suivie. Si la valeur est trouvée, *AndroidPropTracker* possède à une recherche dichotomique sur la liste de commits et recommence tout le processus jusqu'à ce que les valeurs des propriétés suivies soient détectées et enregistrées dans deux ensembles de données : Les changements de propriété et les changements continuels de propriété (voir section If a value is found, Android Prop Tracker binary searches the commit list and restarts this whole process until all values of the tracked property are detected and saved into two datasets : Property changes and continuous property changes (voir section 4.4 articles (i) et (ii)).

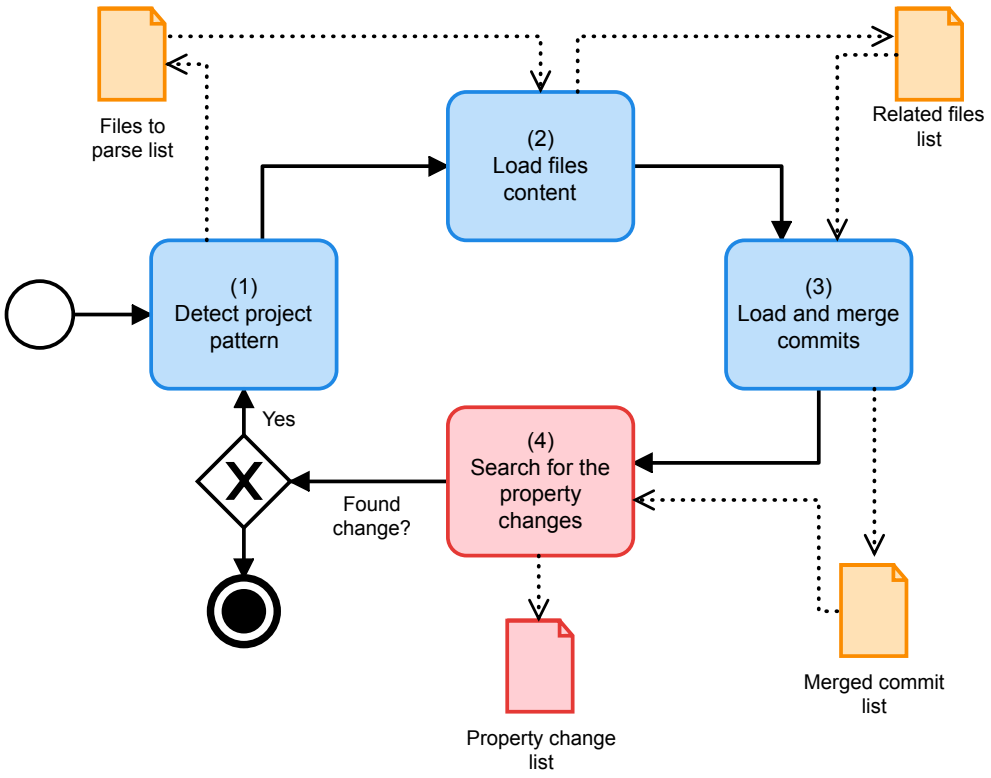

Figure 4.2 – Organigramme - Détection des changements de propriété

Les dépôts Git identifient chaque commit avec un identifiant hash. Les identifiants de Git

Hash sont générés avec l'algorithme SHA-1, qui comporte 40 chiffres hexadécimaux. La limite des commits est la même que le nombre d'identifiants uniques que SHA-1 peut produire, c'est-à-dire 16$\hat{0}$ ou 1,4x10$\hat{4}$8. Avec un nombre de limites de commits aussi élevé, permettant virtuellement l'entrée d'un tableau de n'importe quelle taille, une optimisation est effectuée pour filtrer le tableau de paires de commit entré dans l'algorithme 1.

Bien qu'un tableau contenant toutes les paires de commit puisse être entré dans l'algorithme 1, une optimisation est effectuée en réduisant le tableau d'entrée, en se limitant aux commits qui affectent uniquement les fichiers gradle et leurs fichiers dépendants. L'algorithme 2 optimise l'entrée de l'algorithme 1, en générant un tableau de paires de commits limité aux fichiers gradle et à leurs dépendants uniquement.

Algorithm 2: composeRelevantCommitsList

Input: gitFilePath : piste complète du dernier fichier validé de gradle ; commitPairs : tableau de paires de commits pour le fichier donné, où les premiers sont les identifiants de commit et les secondes sont les valeurs de targetSdkVersion.

Output: commitPairs : tableau de commits affectant uniquement les fichiers Gradle et leurs dépendants.

```
1 fileContent = readFileContent(gitFilePath) while fileContent contains
   'applyfrom :$appliedGitFileName' do
2     appliedGitFileCommitPairs = getCommitPairsForFile(appliedGitFilePath)
3     commitPairs = merge(commitPairs, appliedGitFileCommitPairs)
4     composeRelevantCommitsList(appliedGitFilePath, commitPairs)
5 commitPairs = sortDesc(commitPairs)
6 return commitPairs
```

Mettre en oeuvre l'algorithme 2 : Le tout dernier fichier gradle validé est entré avec son propre tableau de paires de commits ; Lire le contenu du fichier gradle ; Pour chaque incidence de '*applyfrom* :' dans le contenu du fichier gradle entré, charger le fichier appliqué du tableau de paires de commits, Fusionner les paires de commits avec les tableau de paires de commits, et appeler 2 de manière récurrente, passer le fichier appliqué et le tableau de fichiers commits comme paramètres ;

Retourner le tableau de paires de commits classées par ordre décroissant.

4.4 TRAITEMENT DE SORTIE

Chaque structure de données de dépôt a sa liste de modules. Chaque module a son propre ensemble de données des modifications de *targetSdkVersion*. *AndroidPropTracker* itère dans chaque dépôt, projet et module. Il récupère les ensembles de données de liste de modification de propriétés, avec des changements *targetSdkVersion* pour chaque projet. Après la récupération de tous les ensembles de données de liste de modifications de propriétés, l'outil consolide les modifications en deux ensembles de données regroupés en mois ou en trimestres en fonction de l'entrée de l'utilisateur :

(i) Changements de propriété. Chaque changement de valeur de propriété, combien de projets sont passés à quel niveau d'API, au fil du temps.

(ii) Valeurs de propriété continues. Valeurs de propriété, combien de projets se trouvent à quel niveau d'API, au fil du temps.

4.5 COMMENT L'UTILISER

Après avoir téléchargé ou cloné le dépôt git, installé Apache et PHP et effectué la configuration de l'environnement [2], *AndroidPropTracker* sera accessible sur l'url définie.

2. The installation manual is available at http ://github.com/deguilardi/androidproptracker

```
/<account_x>/<repository_x>:<branch_x>:<folder_x>
/<account_y>/<repository_y>:<branch_y>:<folder_y>
...
/<account_n>/<repository_n>:<branch_n>:<folder_n>
```

Listing 4.2 – Liste des modèles de projets. Un dépôt par ligne sans espaces. Les arguments <*branch*> et <*folder*> sont facultatifs.

4.5.1 ENTRÉE

Lorsqu'*AndroidPropTracker* est accessible sur le navigateur Web, l'outil nécessite trois entrées pour commencer la récupération :

(1) Une liste des dépôts ;

(2) La propriété à suivre. Les propriétés *minSdkVersion*, *compileSdkVersion* et *targetSdkVersion* sont actuellement prises en charge ;

(3) Une plage (minimum et maximum) de la propriété suivie.

La listage des projets doit suivre le modèle suivant :

La plage de propriétés définit les valeurs minimale et maximale de la propriété à suivre. *AndroidPropTracker* est testé et actuellement capable de suivre *minSdkVersion*, *targertSdkVersion* et *maxSdkVersion*. Un travail futur pourrait ajouter plus de propriétés ou même permettre à l'outil de suivre n'importe quelle propriété.

La figure 4.3 affiche le formulaire de saisie. Pour effectuer une recherche, l'utilisateur doit entrer une liste de dépôts, sélectionner une propriété à suivre et une plage de propriétés pour le résultat.

Pour effectuer une recherche de modifications, *AndroidPropTracker* récupère la dernière

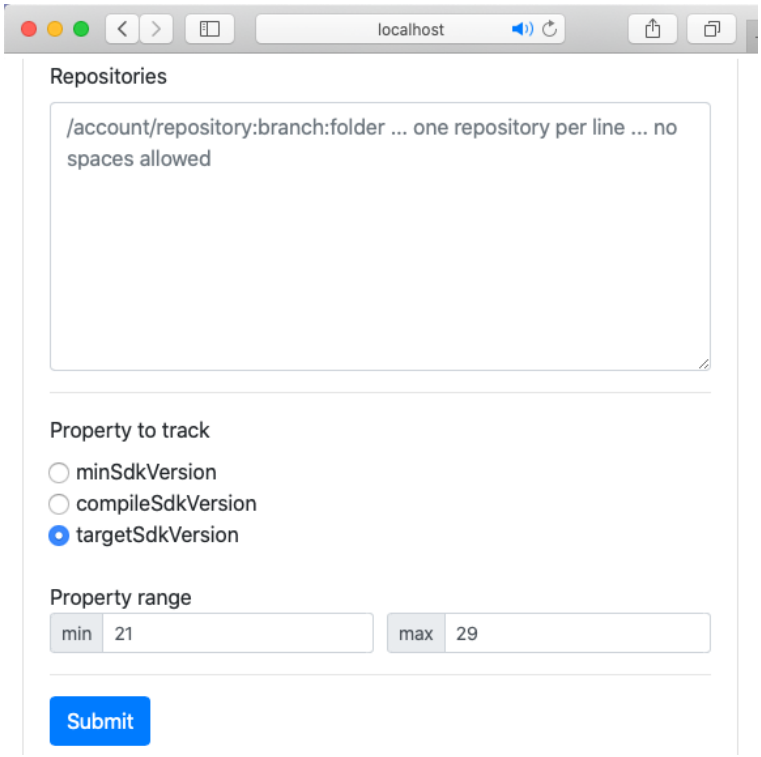

Figure 4.3 – Formulaire de saisie

valeur connue de la propriété suivie. Par la suite, *AndroidPropTracker* récupère une liste de commits du fichier Gradle de chaque module. Ensuite, l'outil effectue une recherche binaire sur les fichiers *Gradle* de chaque module et racine le long de la liste des commits.

4.5.2 SORTIE DES DONNÉES

Notre méthode d'extraction de données examine toute la durée de vie des dépôts. En exploitant cette méthode, nous sommes en mesure de générer des ensembles de données plus détaillés contenant toutes les valeurs qu'une propriété suivie a eues.

AndroidPropTracker fournit un rapport de pipeline avec tous les dépôts filtrés ainsi que les projets avec des modifications détectées dans la propriété suivie (voir figure 4.4). L'outil fournit

également deux ensembles de données comprenant des données quantitatives et qualitatives, ainsi que trois chartes qui ont pour but de faciliter la compréhension des données (voir figures 4.4 et 4.5).

La figure 4.4 illustre le rapport de pipeline où la boîte bleue contient le nombre de dépôts d'entrée, la boîte jaune affiche les dépôts filtrés, la boîte grise contient le nombre de dépôts après la filtration, la boîte verte affiche une liste d'objets qui ont au moins un changement détectée sur la propriété suivie. La boîte rouge rapporte une liste avec aucun changement détecté. La figure 4.4 affiche également l'ensemble des données de tous les changements de la propriété suivie détectés au fil du temps (en mois). La figure 4.4 affiche également un graphique en courbes réalisé sur l'ensemble de données des modifications détectées, dont chaque ligne correspond à une valeur de la propriété suivie, l'axe des x est le temps (en mois) et l'axe des y est le nombre de modifications détectées.

La figure 4.5 affiche un graphique linéaire et un graphique linéaire empilé. Les deux graphiques affichent le nombre de projets avec leur valeur de propriété suivie au fil du temps (en mois). Chaque couleur représente une valeur de la propriété suivie. Ces graphiques contiennent une vue de l'évolution continue de la propriété suivie. Ils montrent le nombre exact de projets ainsi que les valeurs de leur propriété suivie au fil du temps

La sortie, les données illustrées dans les figures 4.4 et 4.5, est ce qui est utilisé pour déterminer la performance de l'état de préparation, comme discuté dans le chapitre 3, section 3.4. La sortie générée par *AndroidPropTracker* peut déterminer la performance de l'état de préparation, d'une application ou d'un groupe d'applications.

4.6 LES SCÉNARIOS D'UTILISATION

Dans cette section, nous discutons des scénarios dans lesquels l'outil *AndroidPropTracker* peut être utile pour collecter et analyser l'évolution d'une propriété sur des projets d'Android. Nous décrivons deux scénarios d'utilisation typiques de l'outil : (1) des chercheurs qui étudient les modifications des propriétés d'Android au fil du temps ; et (2) les développeurs désirant comprendre comment leurs projets évoluent.

Recherches *AndroidPropTracker* est utile pour soutenir les recherches lorsque c'est nécessaire pour suivre et analyser une ou plusieurs propriétés d'Android. Le suivi des propriétés Android peut aider les chercheurs à comprendre l'évolution des projets d'Android ainsi qu'à répondre à différentes questions de recherche. Un exemple d'*AndroidPropTracker* qui soutient les chercheurs est une recherche que j'ai menée dans le but de répondre aux questions de recherche suivantes : *(RQ1)* Les applications sont-elles prêts pour les nouvelles versions d'Android ? ; *(RQ2)* Quels sont les délais permettant de s'adapter aux nouvelles versions d'Android, s'il y en a ? *AndroidPropTracker* nous a supportés afin de répondre à ces deux questions de recherche. Nous avons utilisé cet outil pour suivre et analyser l'évolution de la propriété de *targetSdkVersion* sur 8420 projets. Nous avons analysé les graphiques de la figure 4.5, qui affichent le nombre d'applications qui utilisaient un niveau d'API à un moment précis. Notre résultats ont montré, parmi de nombreuses autres constatations que les applications d'Android sont devenues, au fil de temps, «moins prêtes».

Le développement des applications d'Android Dans ce scénario, nous avons considéré un développeur d'Android prêt à évaluer l'évolution des propriétés de ses applications mobiles. Afin d'effectuer l'évaluation, le développeur peut utiliser *AndroidPropTracker* pour suivre l'évolution de

ses propres applications. Un développeur peut également comparer ses applications avec d'autres applications open source similaires sur Github. Ces deux usages, combinés ou non, permettent aux développeurs d'avoir une vision plus large de l'évolution de leurs applications, ainsi que de l'évolution de la concurrence.

Les ensembles de données générés par l'outil amènent suffisamment de données au développeur, lui permettant d'effectuer une analyse profonde des modifications au fil du temps. Au fil de cette analyse, les mauvaises et bonnes actions sont perçues, donnant lieu à une prise de décision sur les activités et les processus, ce qui mène vers une meilleure qualité de produit et une meilleure correspondance avec le marché.

Le marché du développement d'applications *AndroidPropTracker* donne non seulement des informations qui aident à comprendre une application, mais il est également capable de mesurer l'état de préparation d'une seule application ou d'un groupe d'applications. Après avoir appliqué une technique statistique de régression linéaire (voir la section 3.4 dans le chapitre 3) aux données mesurées, les entreprises de développement d'applications disposent de matériel mathématique indiquant si leur application a tendance à être à jour ou non.

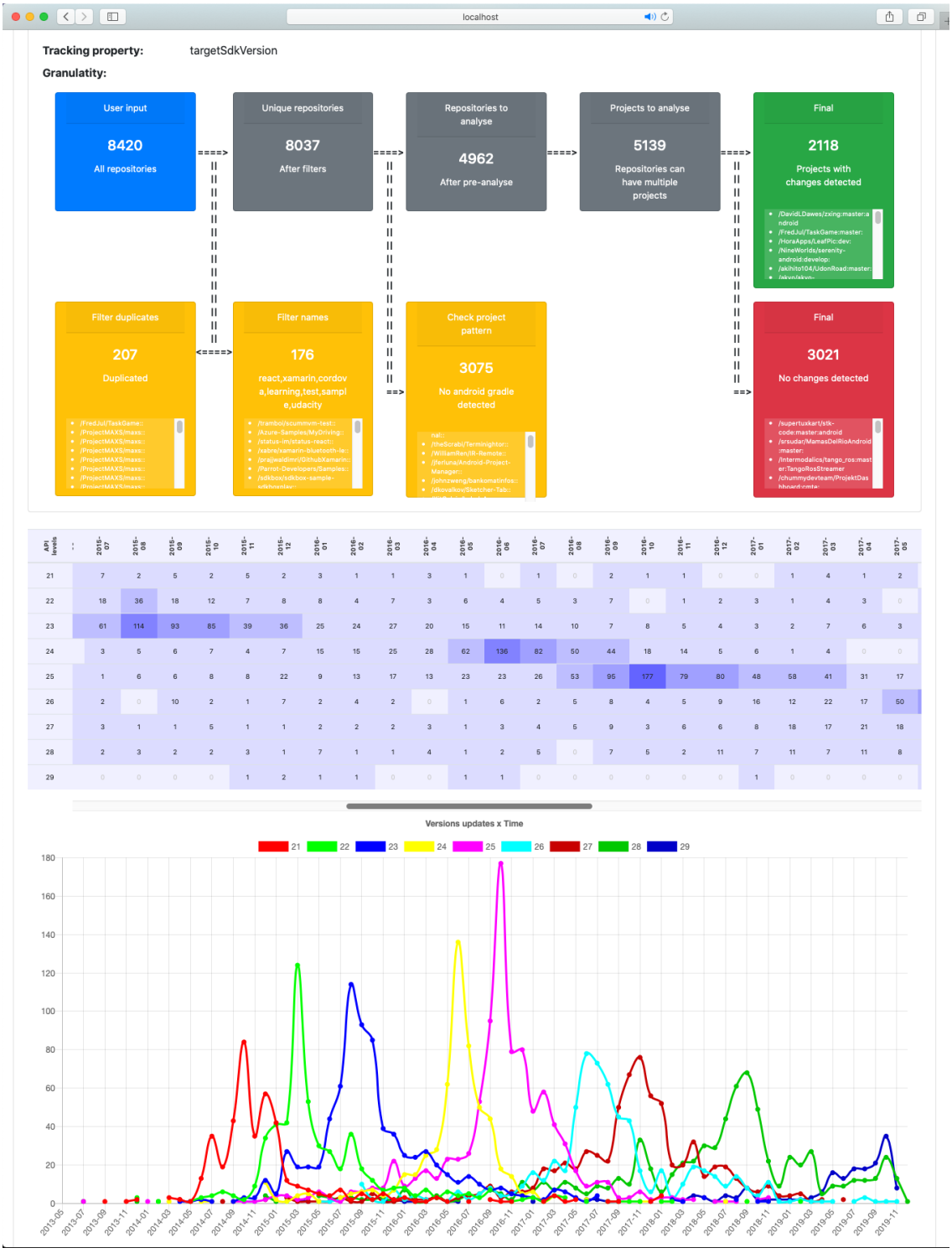

Figure 4.4 – Sortie des données - Rapport de pipeline et changements des propriétés au mois

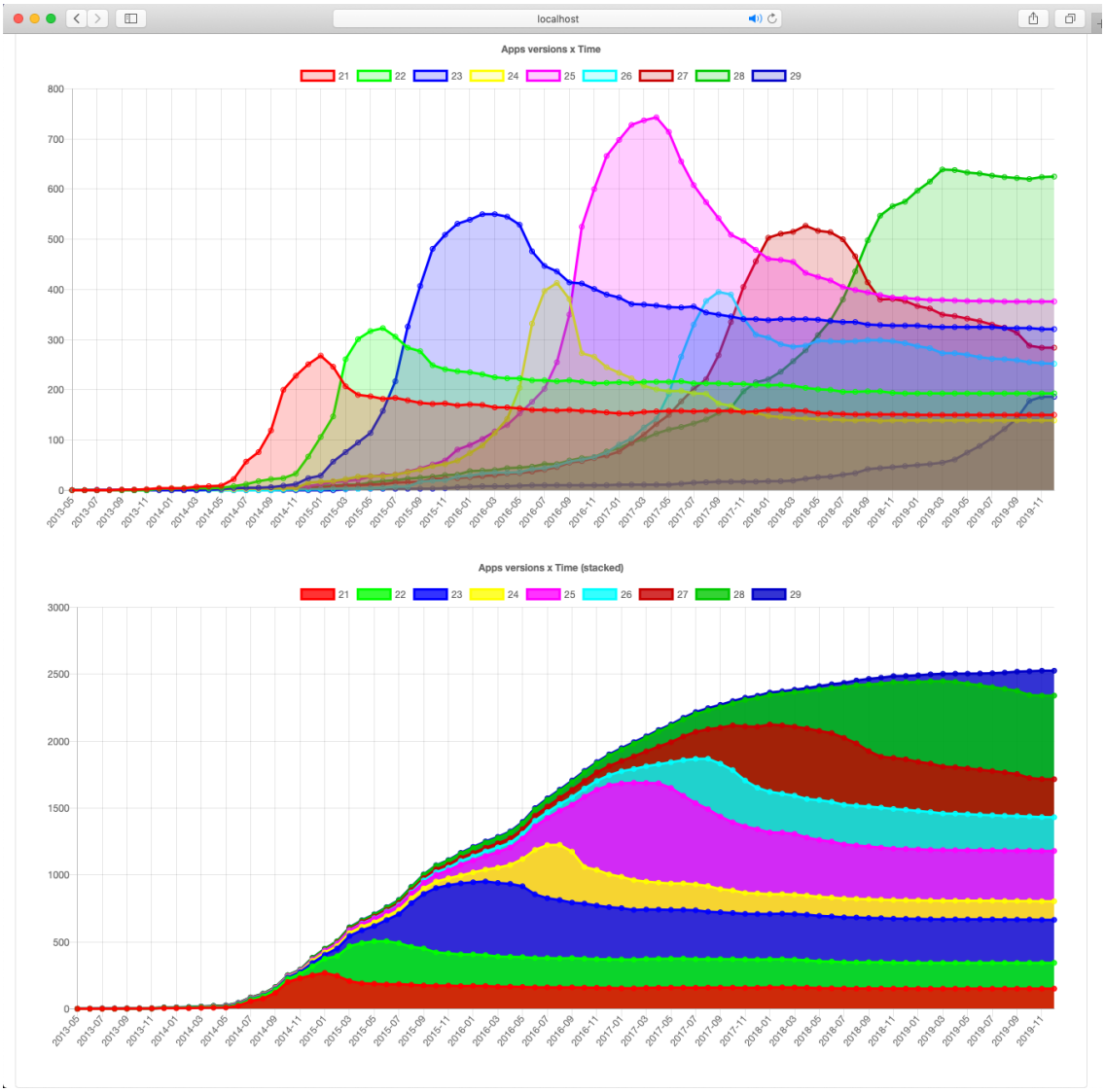

Figure 4.5 – Sortie des données - Changements continuels de la propriété au mois

CHAPITRE 5

L'APPLICATION DE L'APPROCHE

Dans ce chapitre, nous décrivons comment notre approche a été appliquée et testée. De plus, nous allons rendre compte des résultats obtenus au fil des tests. Finalement, nous allons également évaluer les données produites ainsi que les résultats.

5.1 VALIDATION

Afin de valider si l'approche présentée est une bonne façon de suivre des propriété d'un projet d'Android au fil du temps, nous avons collecté et analysé les données. La validation est basée comme si les changements et l'évolution d'une propriété donnée produiraient des données permettant d'éclairer les savoirs et procéder à des conclusions, au fil du temps.

Deux publications ont aussi supporté la validation de notre approche. Dans des publications précédentes (Guilardi *et al.*, 2020b,a), nous avons, avec succès, appliqué notre approche pour répondre aux questions de recherche suivantes : *RQ1* : Les applications sont-elles prêtes pour de nouvelles versions d'Android ? *RQ2* : Quels sont les délais, s'il y en a, pour s'adapter aux nouvelles

versions d'Android ?

Après avoir exécuté *AndroidPropTracker* sur 8420 dépôts et avoir détecté 2118 projets avec des changements de *targetSdkVersion*, les deux sorties ont été copiées sur une fiche de calcul afin de fusionner les résultats de l'outil avec les dates de sortie d'API (Tableau 2.1) :

(i) Changements de propriété (voir l'item (i) dans la section 4.4).

Tableau 5.1 démontre les résultats fusionnés Ça démontre le nombre de changements de *targetSdkVersion* au fil du temps. Chaque mois de sortie d'API est souligné et est représenté par les colonnes soulignées en bleu. La carte thermique est affichée car plus l'arrière-plan est clair, plus la valeur est basse. Les niveaux d'API inférieurs ou égaux à 20 n'ont pas été pris en compte dans cette étude car les données collectées sont insuffisantes.

(ii) Valeurs de propriétés continues (voir l'item (ii) dans la section 4.4).

Figure 5.1 displays merged results Il montre le nombre de projets avec leur *targetSdkVersion* au fil du temps. Chaque mois de sortie de l'API est mis en évidence et représenté par des colonnes surlignées en bleu. La carte thermique est affichée car plus l'arrière-plan est clair, plus la valeur est basse. La figure 5.1 affiche également un graphique constitué des résultats présentés dans le tableau. Le graphique affiche le nombre de projets avec leur targetSdkVersion au fil du temps. Les points colorés sur les lignes indiquent la date de sortie de chaque API, les losanges noirs indiquent les sommets de chaque API. Nous avons regroupé les valeurs de tous les niveaux d'API inférieurs ou égaux à 20 pour avoir une base, bien que ces versions n'aient pas été prises en compte dans cette étude puisque les données collectées sont insuffisantes. Les deux lignes droites sont des régressions linéaires (voir chapitre 3, section 3.4).

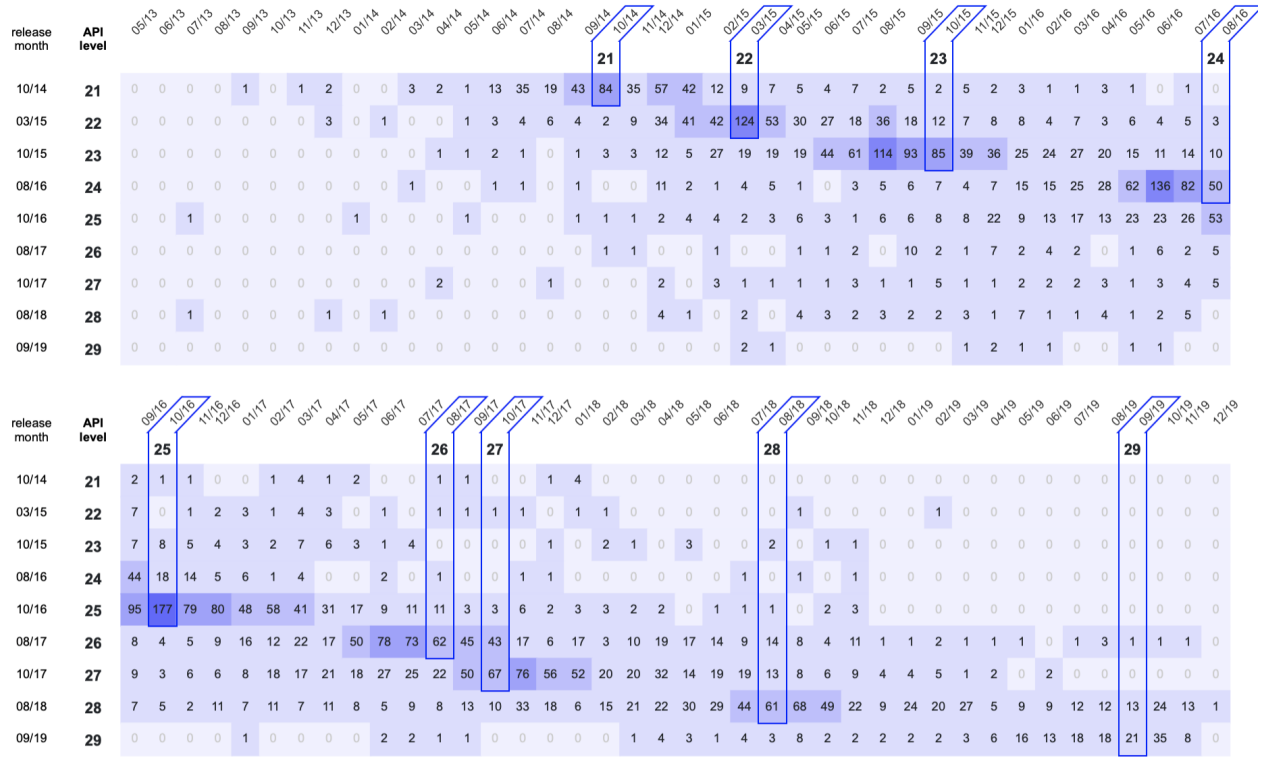

Tableau 5.1 – Changements de targetSdkVersion au fil du temps, avec les points forts de chaque sortie d'API.

Après avoir organisé les ensembles de données fournis par *AndroidPropTracker* , nous avons ensuite commencé à les analyser pour arriver à des résultats.

5.2 RÉSULTATS DE VALIDATION

Nous avons calculé des totaux, généré des sous-ensembles e des graphiques sur les ensembles de données d'origine [1]. Après avoir entré 8420 dépôts sur *AndroidPropTracker* , 207 dépôts ont été filtrés pour être dupliqués, 176 dépôts ont été filtrés car ils n'étaient pas conformes aux critères de sélection (voir 4.2), 3075 dépôts ont été organisés de manière à ce qu'*AndroidPropTracker* ne puisse pas être en mesure de détecter la propriété suivie. Sur les 4962 dépôts restants, 5139 projets ont été

1. *AndroidPropTracker* est disponible sur Github : https ://github.com/deguilardi/androidPropTracker

trouvés, 2118 avaient des changements de *targetSdkVersion* qui ont été détectés et 3021 n'en avaient pas.

Nous avons trouvé des valeurs aberrantes qui visaient des niveaux d'API trop anciens, trop récents ou encore inexistants. Ces valeurs aberrantes sont visibles dans les ensembles de données des schémas 5.1 et 5.1.

5.2.1 L'ÉTAT DE PRÉPARATION EST MESURABLE

Une des conclusions de cette étude est que l'état de préparation des applications d'Android peut être mesuré. Le niveau de préparation des applications pour les nouvelles versions d'Android, ou l'état de préparation, est mesurable en examinant les valeurs de la variable *targetSdkVersion*, dans un ensemble de projets d'application d'Android. Pour un moment et une version d'Android spécifique, il est possible d'obtenir des résultats mesurables, tels que le pourcentage exact d'applications prêtes pour la version donnée, pas prêtes, prêtes pour les versions précédentes ou prêtes pour une future version d'Android.

5.2.2 L'ÉTAT DE PRÉPARATION EST COMPARABLE

Une autre conclusion de cette étude est que l'état de préparation peut également être comparé. En examinant l'état de préparation du point de vue des versions Android, il est également possible de comparer l'état de préparation. Comme vu dans la section 5.2, figure 5.1, l'évolution de l'adhésion à une version n'est pas seulement mesurable, elle est également comparable avec d'autres versions.

La figure 5.2 contient un exemple de comparaison de l'état de préparation. Chaque ligne du

tableau, représentée par des lignes dans le graphique, contient des données comparables.

5.2.3 L'ÉTAT DE PRÉPARATION EST PRÉVISIBLE

Cette étude conclut également que non seulement l'état de préparation est mesurable et comparable, mais qu'il est également prévisible. Du point de vue de la version Android, il est également possible de mesurer l'état de préparation dans un laps de temps précis. Après avoir appliqué une technique statistique de régression linéaire, (chapitre 3, section 3.4) sur les changements de l'état de préparation au fil du temps (voir figure 5.1), nous avons également conclu que l'état de préparation est prévisible.

release month	API level	05/13	06/13	07/13	08/13	09/13	10/13	11/13	12/13	01/14	02/14	03/14	04/14	05/14	06/14	07/14	08/14	09/14	10/14	11/14	12/14	01/15	02/15	03/15	04/15	05/15	06/15	07/15	08/15	09/15	10/15	11/15	12/15	01/16	02/16	03/16	04/16	05/16	06/16	07/16	08/16
			18			20												21					22								23										24	
various	<= 20	1	3	14	16	19	34	43	43	46	51	56	57	64	68	79	91	92	77	73	71	75	80	85	89	90	92	95	97	95	96	97	98	98	98	98	99	101	101	103	102
10/14	21	0	0	0	0	1	1	2	4	4	4	7	8	9	22	57	76	119	200	228	251	268	246	207	190	187	182	184	179	174	172	173	169	171	170	165	165	163	160	160	159
03/15	22	0	0	0	0	0	0	0	3	3	4	4	4	5	8	12	18	22	24	33	67	106	147	261	301	317	323	306	284	277	249	241	237	235	231	225	223	223	219	219	217
10/15	23	0	0	0	0	0	0	0	0	0	0	0	1	2	4	5	5	6	9	12	24	29	57	76	95	114	158	217	326	407	481	509	531	539	550	550	545	529	476	447	436
08/16	24	0	0	0	0	0	0	0	0	0	0	1	1	1	2	3	3	4	4	4	15	17	18	22	27	28	28	31	36	41	48	52	59	74	89	114	142	204	332	397	413
10/16	25	0	0	1	1	1	1	1	2	2	2	2	3	3	3	3	4	5	6	8	12	16	18	21	27	30	31	37	43	51	59	81	90	102	117	130	153	176	202	255	
08/17	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	2	2	3	3	3	4	5	7	7	17	19	20	27	29	33	35	35	36	42	44	49	
10/17	27	0	0	0	0	0	0	0	0	0	0	2	2	2	2	3	3	3	3	5	5	8	9	10	11	12	15	16	17	22	23	24	26	28	30	33	34	37	41	46	
08/18	28	0	0	1	1	1	1	2	2	3	3	3	3	3	3	4	4	4	4	8	9	9	11	11	15	18	20	23	25	27	30	31	38	39	40	44	45	47	52	52	
09/19	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	3	3	3	3	3	3	4	6	7	8	8	8	9	10	10	10	

release month	API level	09/16	10/16	11/16	12/16	01/17	02/17	03/17	04/17	05/17	06/17	07/17	08/17	09/17	10/17	11/17	12/17	01/18	02/18	03/18	04/18	05/18	06/18	07/18	08/18	09/18	10/18	11/18	12/18	01/19	02/19	03/19	04/19	05/19	06/19	07/19	08/19	09/19	10/19	11/19	12/19
			25										26		27									28														29				
various	<= 20	100	100	100	101	99	100	101	101	101	103	103	102	102	102	103	102	102	102	102	102	102	103	103	103	104	104	104	103	103	103	103	103	103	103	103	102	102	102	102	102
10/14	21	160	158	157	155	153	153	156	157	158	158	157	158	158	158	156	157	160	160	159	158	153	153	152	151	151	151	151	151	150	150	150	150	150	150	150	150	150	150	150	150
03/15	22	219	216	213	214	215	214	216	216	216	217	213	213	213	212	212	210	209	210	208	204	201	200	196	196	197	197	194	193	193	193	193	193	193	193	193	193	193	193	193	193
10/15	23	414	412	401	390	384	371	370	368	365	364	366	354	350	346	341	341	339	341	341	341	340	337	335	335	330	329	328	328	328	326	325	325	325	325	325	323	323	323	321	321
08/16	24	381	273	266	245	234	223	208	201	197	198	193	192	173	168	156	154	148	146	144	143	142	141	140	139	140	138	139	139	139	139	139	139	139	139	139	139	139	139	139	139
10/16	25	350	525	600	666	698	728	737	743	714	655	608	574	542	509	497	479	461	459	455	433	425	418	405	399	394	389	384	383	381	379	379	378	377	377	377	376	376	376	376	376
08/17	26	57	61	66	75	91	103	125	142	192	266	330	377	395	390	343	310	304	291	286	288	298	297	296	297	299	299	297	293	287	283	273	273	270	265	262	261	259	255	253	252
10/17	27	55	58	64	69	77	94	111	132	150	177	202	221	269	335	405	456	503	511	515	527	517	514	500	466	414	380	381	377	367	362	350	347	342	337	330	324	314	288	284	284
08/18	28	59	64	66	77	84	95	102	113	121	126	133	141	154	164	197	215	221	236	257	279	309	338	380	436	498	547	566	575	597	615	639	638	633	631	627	624	622	620	624	625
09/19	29	10	10	10	10	11	11	11	11	11	13	15	16	17	17	17	17	18	18	19	23	26	27	31	34	42	44	46	48	50	52	55	61	75	88	104	122	143	178	186	186

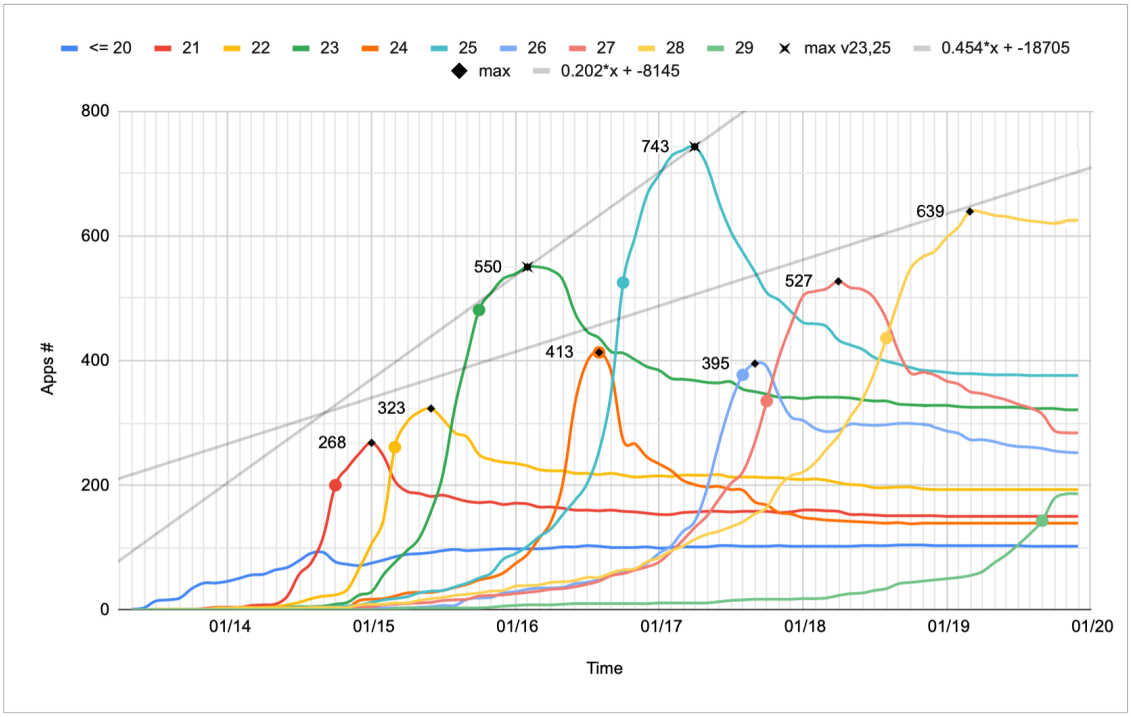

Figure 5.1 – Valeurs de compatibilité ascendante des applications au fil du temps, avec des mises en évidence sur chaque version d'API. Les deux lignes droites du graphique sont des régressions linéaires (voir 3, section 3.4)

API level	months before release												release month	months after release															
	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	24	36	48	60
21	1	2	4	4	4	7	8	9	22	57	76	119	200	228	251	268	246	207	190	187	182	184	179	174	172	158	158	151	150
22	4	4	5	8	12	18	22	24	33	67	106	147	261	301	317	323	306	284	277	249	241	237	235	231	225	216	208	193	-
23	9	12	24	29	57	76	95	114	158	217	326	407	481	509	531	539	550	550	545	529	476	447	436	414	412	346	329	323	-
24	36	41	48	52	59	74	89	114	142	204	332	397	413	381	273	266	245	234	223	208	201	197	198	193	192	139	139	-	-
25	51	59	81	90	102	117	130	153	176	202	255	350	525	600	666	698	728	737	743	714	655	608	574	542	509	389	376	-	-
26	49	57	61	66	75	91	103	125	142	192	266	330	377	395	390	343	310	304	291	286	288	298	297	296	297	261	-	-	-
27	58	64	69	77	94	111	132	150	177	202	221	269	335	405	456	503	511	515	527	517	514	500	466	414	380	288	-	-	-
28	141	154	164	197	215	221	236	257	279	309	338	380	436	498	547	566	575	597	615	639	638	633	631	627	624	-	-	-	-
29	42	44	46	48	50	52	55	61	75	88	104	122	143	178	186	186	-	-	-	-	-	-	-	-	-	-	-	-	

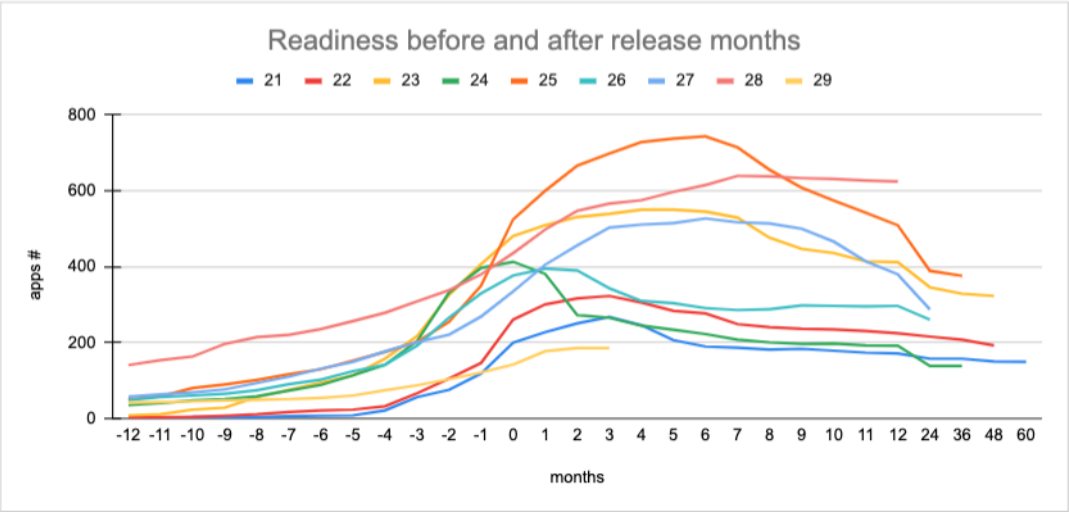

Figure 5.2 – L'état de réparation avant et après les mois de sortie d'API. Chaque ligne représente une version d'API. Chaque colonne représente des mois avant et après la sortie des API. La colonne en surbrillance représente le mois de sortie de chaque version d'API. Le graphique est une représentation graphique du tableau directement au-dessus.

CHAPITRE 6

DISCUSSIONS

Notre approche présente des techniques pour mesurer à quel point les applications sont prêtes pour les nouvelles versions d'Android dans une échelle d'intervalle significative (Briand *et al.*, 1996). En évaluant l'état de préparation à l'échelle établie, cette approche présente des résultats qui montrent que les valeurs des paramètres de *targetSdkVersion* au fil du temps, lorsqu'ils sont analysés en profondeur, révèlent que l'état de préparation n'est pas seulement mesurable, il est également comparable et prévisible. Cette étude présente également des résultats révélant des implications qui affectent le champ de recherche sur le développement d'Android, les développeurs, ainsi que des implications qui pointent vers les politiques de Google et les utilisateurs d'Android.

6.1 MEASURABILITÉ

L'état de préparation des applications d'Android est mesurable. Le niveau de préparation des applications pour les nouvelles versions d'Android, ou l'état de préparation, est mesurable en examinant les valeurs de la variable *targetSdkVersion*, dans un ensemble de projets d'application d'Android. Pour un temps précis et une version d'Android spécifique, il est possible d'obtenir des

résultats mesurables, tels que le pourcentage exact d'applications prêtes pour la version donnée, pas prêtes, prêtes pour les versions précédentes ou prêtes pour une future version d'Android.

Pour créer une échelle d'intervalle significative (Briand *et al.*, 1996), nous avons proposé que la mesure de l'état de préparation au fil du temps soit sous forme de durée. Mesurant combien de retard ou combien d'avance, au lieu d'une simple échelle nominale sous forme de oui ou non. En mesurant l'état de préparation dans une échelle d'intervalle, il est possible de répondre à des questions significatives telles que "À quel point une application est-elle prête ?" ou "Quelle est la variation de l'état de préparation d'une application au fil du temps ?".

L'état de préparation est mesurable de différentes façons, individuellement ou par groupes d'applications. Pour analyser une seule application, les variations de l'état de préparation sont mesurées au fil du temps. (voir figures 3.3 and 3.4). L'état de préparation d'un groupe d'applications est également mesurable, comme démontré dans le tableau 5.1 et les figures 5.1 et 5.2.

6.2 COMPARABILITÉ

L'état de préparation est comparable. En regardant l'état de préparation d'une application, on peut mesurer l'état de préparation d'une version Android qui varie d'une à l'autre. Ce phénomène rend l'état de préparation comparable, en examinant la disponibilité d'une seule application dans le temps.

L'état de préparation est également comparable entre les différentes applications. La mesure de l'état de préparation de différentes applications grâce à notre approche le rend comparable en permettant de comparer à quel point les applications étaient, sont et seront prêtes entre elles.

De plus, l'état de préparation est également comparable du point de vue des versions d'Android. Différentes versions d'Android ont des performances d'état de préparation différentes. Comme montré dans le schéma 5.1, différentes versions d'Android peuvent avoir une différente adhésion. La figure 5.1 affiche les données d'Android 6.0 (niveau d'API 23) et 7.1 (niveau d'API 25) ayant de plus hautes adhésions que les autres.

Une autre discussion tirée de cette étude est que l'état de préparation peut également être comparé. En examinant l'état de préparation du point de vue des versions d'Android, il est également possible de comparer l'état de préparation. Comme vu dans la section 5.2, la figure 5.1,l'évolution de l'adhésion vers une version n'est pas seulement mesurable, elle est aussi comparable avec d'autres versions

6.3 PRÉVISIBILITÉ

La mesurabilité et la comparabilité de l'état de préparation donnent lieu à la prévisibilité. La prévisibilité est définie comme «l'état de savoir à quoi ressemble quelque chose, quand quelque chose va se passer» (Cambridge Dictionary Online, 2020). Grâce à l'utilisation de techniques statistiques, telles que la régression linéaire, tout parti intéressé par l'étude de l'état de préparation des applications Android est capable de prédire l'état de préparation d'une application ou d'un groupe d'applications. Le coefficient de régression β, détermine si les applications ont tendance à être à jour ou non. Les coefficients de régression positifs indiquent que la préparation est de plus en plus en avance sur les versions d'Android, tandis que les coefficients négatifs indiquent que la préparation est de moins en moins en avance (ou de plus en plus en retard) sur les versions d'Android.

Selon Wohlin *et al.* (2000), les expérimentations sont importantes pour tester une approche

et en particulier la capacité prédictive de l'approche. Des études et des observations empiriques ont déjà été effectuées en utilisant cette approche (Guilardi *et al.*, 2020b,a), et les résultats ont montré que l'état de préparation était prévisible. Comme (Guilardi *et al.*, 2020b) a conclu que, au début de 2020, 4.41% des applications étaient destinées à fonctionner sur des Androids qui datent de plus de six ans. Également au début de 2020, 94,55% des applications n'étaient pas déclarées compatibles avec la dernière version d'Android (niveau d'API 29). Ces expériences (Guilardi *et al.*, 2020b,a) prédisent que les applications Android ont tendance à avoir des coefficients de régression négatifs et à être moins prêtes avec le temps.

6.4 L'IMPACT DU DÉVELOPPEMENT D'ANDROID

La documentation officielle affirme que : «Pour maintenir votre application avec chaque version d'Android, vous devez augmenter la valeur de *targetSdkVersion* pour qu'elle corresponde au dernier niveau d'API, puis tester minutieusement votre application sur la version de plate-forme correspondante» (Android Developers, ndc). Le texte est une suggestion indiquant que les développeurs "devraient" garder leurs applications à jour. Cette brève suggestion est la meilleure instruction que nous ayons trouvée, alertant les développeurs sur l'importance d'effectuer la compatibilité ascendante sur leurs applications.

L'adaptation d'une application pour prendre en charge une nouvelle version d'Android nécessite plus qu'une simple augmentation de la valeur de *targetSdkVersion* et un test de l'application. Faire correspondre *targetSdkVersion* avec le dernier niveau d'API indique à Android Studio, Gradle et Play Store que l'application est en cours de développement, de test, de compilation, de déploiement et de distribution correspondant au niveau d'API spécifié. Bien que les adaptations puissent également nécessiter des changements de code.

Lorsqu'une nouvelle version d'Android est accompagnée de changements au niveau de son comportement, ces derniers étant de profonds changements d'OS, les applications doivent s'adapter en modifiant également leurs propres comportements. Par exemple, Android 9 (niveau d'API 28) a restreint l'accès à la caméra et au microphone en mode d'arrière-plan (Android Developers, nda). À partir de cette version, aucune application ne pourra accéder à la caméra ou au microphone d'un appareil Android si l'application est en mode arrière-plan. Ce changement de comportement vise à ajouter de la sécurité à Android OS, bien qu'il nécessite un changement majeur dans les applications de surveillance, car à partir d'Android 9, ils ne pourront plus suivre la caméra ni le microphone.

Rendre prête une application pour une nouvelle version d'Android peut être aussi simple que de simplement changer *targetSdkVersion*, faire de petites fixations et adaptations de code, effectuer des changements architecturaux importants de logiciels ou même restructurer un modèle commercial. Dans tous les cas, les adaptations nécessitent une planification et plusieurs tests.

6.5 LES IMPACTS SUR LES UTILISATEURS D'ANDROID

Selon GlobalStats (2019), est installé sur plus du trois quarts des téléphones intelligents mondialement. En septembre 2019, le mois de sortie de la dernière mise à jour d'Android (Android 10, niveau d'API 29), seulement 5.45% des applications étaient assurées par leur développeur d'être compatibles avec cette nouvelle version. Le fait que 94.55% des applications n'ont pas la compatibilité ascendante pour la dernière version d'Android met à l'évidence un volume sans précédent d'applications qui pourraient être susceptibles d'arrêter de fonctionner ou encore d'exposer les utilisateurs à différents risques (Mutchler *et al.*, 2016). Google a fait l'effort d'accroître l'adhésion aux nouvelles versions d'Android en limitant la publication ou la mise à jour des applications sur le Play Store si elles ne sont pas compatibles avec une version d'Android récente et prédéfinie (Android

Developers Blog, 2019a, 2017b).

Il n'existe aucun mécanisme permettant aux utilisateurs d'Android d'être réellement conscients des risques potentiels auxquels ils font face en utilisant leurs appareils. Les utilisateurs ne sont pas en mesure de savoir si les applications sont adaptées pour fonctionner sur leurs appareils à partir du moment où les applications sont téléchargées du Play Store, jusqu'à des années plus tard, en gardant sur leurs appareils des applications n'étant plus à jour, après la mise à jour du système OS vers des versions plus récentes.

Actuellement, le Play Store de Google indique qu'un "appareil n'est pas compatible" avec une application, en interdisant l'installation, lorsque l'application nécessite une version minimale du système OS ou un matériel spécifique qui n'est pas fourni par l'appareil en question. Par exemple, une application qui déclare qu'elle fonctionne avec un minimum de niveau d'API de 19 (Kitkat 4.4) ne fonctionnera certainement pas sur les niveaux d'API antérieurs à Kitkat 4.4. Ou bien, une application qui déclare avoir besoin d'une caméra ne fonctionnera pas sur les appareils sans caméra. Ces deux avertissements d'incompatibilité sont basés sur le fait que le matériel n'est pas compatible ou que le système OS est trop ancien pour l'application (compatibilité descendante, section 2.2). Les utilisateurs ne sont jamais avertis lorsque le système OS est trop récent pour l'application. Nous suggestons une **indication de compatibilité** sur la page Play Store de chaque application, indiquant si une application est assurée d'être compatible (section 2.2) avec la version ultérieure d'Android sur l'appareil de l'utilisateur avant le téléchargement ainsi que l'installation de l'application.

Une fois les applications déjà installées, un appareil peut être mis à jour avec les nouvelles versions d'Android. Il n'y a pas de limite quant au nombre de versions qu'un appareil peut mettre à jour. Les applications peuvent tout de même explorer les failles de sécurité et exposer les utilisateurs à des risques (Mutchler *et al.*, 2016).

6.6 L'IMPACT DES POLITIQUES DE GOOGLE

Afin de minimiser l'effet du manque au niveau de l'état de préparation, chaque version d'Android a un mode de compatibilité qui imite la conduite des anciennes versions. Cette compatibilité descendante permet aux applications qui sont faites pour fonctionner sur les anciennes versions d'Android de fonctionner sur les nouvelles versions. Google a également instauré de nouvelles politiques depuis 2017. Chaque année, il y a une date limite pour que les applications soient adaptées et prêtes pour l'un des derniers niveaux d'API défini préalablement. Google s'assure également d'appliquer des sanctions aux applications qui ne sont pas prêtes à la date d'échéance. Les trois *Restriction de publication sur Google Play* imposées jusqu'à présent (Android Developers Blog, 2017b, 2019a) ont donné quelques mois aux développeurs pour rendre leurs applications prêtes.

Ces politiques imposent aux applications d'être prêtes pour une version d'Android prédéterminée avant même que cette dernière soit publiée sur le Play Store. Cela veut dire que Google a peut-être remarqué une baisse au niveau de l'état de préparation, comme une étude précédente (Guilardi *et al.*, 2020b) a conclu que moins de 5% des applications d'Android étaient prêtes pour Android 10 (niveau d'API 29), au début de l'année 2020.

CHAPITRE 7

CONCLUSIONS ET FUTUR TRAVAIL

L'existence de plusieurs API d'Android fait partie d'un phénomène connu sous le nom de fragmentation (Han *et al.*, 2012; Wu *et al.*, 2013; Zhou *et al.*, 2014). La fragmentation est définie comme ayant différents appareils, provenant de divers fabricants, avec différentes spécifications, couvrant différentes versions d'Android. La fragmentation excessive a causé des problèmes au niveau de la planification, du développement et des essais des applications d'Android (Joorabchi *et al.*, 2013).

Google et les développeurs font des efforts de manière à minimiser les problèmes de fragmentation, en s'assurant que les applications fonctionneront de concert avec les nouvelles versions d'Android, tout comme les anciennes. Google a l'objectif de garantir que les nouveaux Systèmes d'exploitation d'Android sont compatibles avec les applications étant faites pour fonctionner sur les anciennes versions d'Android (compatibilité descendante). D'un autre côté, les développeurs se concentrent sur le fait de garantir que leurs applications fonctionnent avec les nouvelles versions d'Android bientôt disponibles (compatibilité ascendante). Le rôle du développeur dans la gestion de la compatibilité des nouvelles versions d'Android est le point central de cette étude.

Dans cette recherche, nous proposons une approche d'évaluation de l'état de préparation d'Android. Notre approche débute par définir ce qu'est l'état de préparation. Après avoir défini l'état de préparation, nous avons utilisé de pratiques d'analyse comparative pour définir les métriques, extraire les mesures de l'état de préparation et déterminer la performance de l'état de préparation.

Les métriques sont définies comme étant l'état de préparation au fil du temps, ayant une liste des versions d'Android comme échelle, dans laquelle une application est comparée à une autre. Les mesures sont effectuées via un algorithme de détection des changements de l'état de préparation, chevauchant les données de l'état de préparation sur l'échelle des sorties de versions d'Android.

Pour déterminer les performances de l'état de préparation, une technique statistique de régression est appliquée aux données collectées de l'état de préparation. Le coefficient de régression détermine les performances de l'état de préparation. Les coefficients positifs indiquent que les applications ont tendance à être prêtes avant les nouvelles versions d'Android.

Après avoir défini les métriques, les mesures et les performances de l'état de préparation, nous avons développé un outil appelé *AndroidPropTracker* , qui met en oeuvre l'algorithme de détection des changements de l'état de préparation (voir algorithme 1, chapitre 3, section 3), et instaure son propre algorithme d'optimisation (voir l'algorithme 2).

Pour évaluer notre approche, Nous avons généré un ensemble de données avec des applications correspondant aux critères de sélection en exploitant un outil nommé *AndroidTimeMachine*, où 8420 référentiels ont été sélectionnés. Après la sélection des projets, Nous avons exécuté *AndroidPropTracker* avec la liste des 8420 applications comme entrée de données. L'outil a extrait des données indiquant avec quelles versions d'Android les applications sélectionnées via *AndroidTimeMachine* ont été configurées pour fonctionner, tout au long de leur vie.

Nous avons effectué une analyse sur les données collectées via toolname sur les résultats qui ont montré que le niveau de préparation des applications pour les nouvelles versions d'Android, ou l'état de préparation, est mesurable, comparable et prévisible. Les mesures, comparaisons et prédictions sont exécutées en recherchant et en comparant un ensemble de variables dans le fichier *build.gradle* , la *targetSdkVersion*.

Il y a encore beaucoup de travail à faire. Nous considèrons que, malgré la baisse du taux d'état de préparation, il y a un récent changement au niveau du comportement des applications étants mises à jour au niveau 28 d'API. Un travail futur pourrait enquêter sur ce qui est exactement arrivé à cette version. De plus, un travail futur pourrait également effectuer une extraction et une analyse de données en continu, dans lesquelles non seulement la compatibilité ascendante serait suivie, mais d'autres paramètres tels que la prise en charge de différentes tailles d'écran et aspects de différents appareils. Un travail futur pourrait également détecter et classer les applications dans des catégories, telles que maintenues, non maintenues, open-source et propriétaires.

Cette approche pourrait évoluer vers des applications plus spécifiques. Les études du cycle de vie des applications, les comparaisons de marché et l'analyse de la qualité sont quelques exemples permettant d'illustrer un potentiel de développer et de faire évoluer cette étude vers des applications en particulier. Notre approche pourrait être développée pour acquérir une compréhension plus précise de domaines spécifiques et de paramètres définis.

L'approche présentée pourrait aussi être poussée vers un aspect plus large. Plus large dans le sens où elle viserait l'exploration d'autres variables, même celles qui ne sont pas directement liées aux niveaux d'API des versions d'Android. Plus vaste dans l'exploration des systèmes d'exploitations d'Android, mais également de ceux d'autres appareils. Notre approche pourrait évoluer, permettant de mesurer les changements et l'évolution de n'importe quel logiciel et système d'exploitation.

BIBLIOGRAPHIE

Alabaster, J. (2013). Android founder : We aimed to make a camera os. Repéré le 11 février 2020 à https ://www.pcworld.com/article/2034723/android-founder-we-aimed-to-make-a-camera-os.html

Anand, G. & Kodali, R. (2008). Benchmarking the benchmarking models. *Benchmarking : An International Journal*, *15*(3), 257–291. doi :10.1108/14635770810876593

Android Blog (2013). Introducing android 4.3, a sweeter jelly bean. Repéré le 9 novembre 2019 à https ://android.googleblog.com/2013/07/introducing-android-43-sweeter-jelly.html

Android Blog (2015). Get ready for the sweet tast of android 6.0 marshmallow. Repéré le 8 novembre 2019 à https ://android.googleblog.com/2015/10/get-ready-for-sweet-taste-of-android-60.html

Android Developers (2019). Distribution dashboard. Repéré le 24 novembre 2019 à https ://developer.android.com/about/dashboards

Android Developers (2020). Apps Manifest Overview. Repéré le 22 octobre 2020 à https ://developer.android.com/guide/topics/manifest/manifest-intro

Android Developers (n.d.a). Android 9 - behavior changes : all apps. Repéré le 11 octobre 2020 à https ://developer.android.com/about/versions/pie/android-9.0-changes-all

Android Developers (n.d.b). Configure your build. Repéré le 9 novembre 2019 à https ://developer.android.com/studio/build

Android Developers (n.d.c). <uses-sdk>. Repéré le 9 novembre 2019 à https ://developer.android.com/guide/topics/manifest/uses-sdk-element.html

Android Developers (n.d.d). Versioning your app. Repéré le 24 octobre 2020 à https ://developer.android.com/studio/publish/versioning

Android Developers Blog (2008). Announcing the android 1.0 sdk, release 1. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2008/09/announcing-android-10-sdk-release-1.html

Android Developers Blog (2009a). Android 1.1 sdk, release 1 now available. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2009/02/android-11-sdk-release-1-now-available.html

Android Developers Blog (2009b). Android 1.5 is here ! Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2009/04/android-15-is-here.html

Android Developers Blog (2009c). Android 1.6 sdk is here. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2009/09/android-16-sdk-is-here.html

Android Developers Blog (2009d). Android sdk updates. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2009/12/android-sdk-updates.html

Android Developers Blog (2009e). Announcing android 2.0 support in the sdk ! Repéré le 8 novembre

2019 à https ://android-developers.googleblog.com/2009/10/announcing-android-20-support-in-sdk.html

Android Developers Blog (2010a). Android 2.1 sdk. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2010/01/android-21-sdk.html

Android Developers Blog (2010b). Android 2.2 and developers goodies. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2010/05/android-22-and-developers-goodies.html

Android Developers Blog (2010c). Android 2.3 platform and updated sdk tools. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2010/12/android-23-platform-and-updated-sdk.html

Android Developers Blog (2011a). Android 2.3.3 platform, new nfc capabilities. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2011/02/android-233-platform-new-nfc.html

Android Developers Blog (2011b). Android 3.1 platform, new sdk tools. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2011/05/android-31-platform-new-sdk-tools.html

Android Developers Blog (2011c). Android 3.2 platform and updated sdk tools. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2011/07/android-32-platform-and-updated-sdk.html

Android Developers Blog (2011d). Android 4.0 platform and updated sdk tools. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2011/10/android-40-platform-and-updated-sdk.html

Android Developers Blog (2011e). Android 4.0.3 platform and updated sdk tools. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2011/12/android-403-platform-and-updated-sdk.html

Android Developers Blog (2011f). Final android 3.0 platform and updated sdk tools. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2011/02/final-android-30-platform-and-updated.html

Android Developers Blog (2012a). Introducing android 4.1 (jelly bean) preview platform, and more. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2012/06/introducing-android-41-jelly-bean.html

Android Developers Blog (2012b). Introducing android 4.2, a new and improved jelly bean. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2012/11/introducing-android-42-new-and-improved.html

Android Developers Blog (2013). Android 4.4 kitkat and updated developer tools. Repéré le 9 novembre 2019 à https ://android-developers.googleblog.com/2013/10/android-44-kitkat-and-updated-developer.html

Android Developers Blog (2014). What's new in android 5.0 lollipop. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2014/10/whats-new-in-android-50-lollipop.html

Android Developers Blog (2015). Android 5.1 lollipop sdk. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2015/03/android-51-lollipop-sdk.html

Android Developers Blog (2016a). Now available : Android 7.1 developer preview. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2016/10/android71-dev-preview-available.html

Android Developers Blog (2016b). Taking the final wrapper off of android 7.0 nougat. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2016/08/taking-final-wrapper-off-of-nougat.html

Android Developers Blog (2017a). Android 8.1 developer preview. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2017/10/android-81-developer-preview.html

Android Developers Blog (2017b). Improving app security and performance on google play for years to come. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2017/12/improving-app-security-and-performance.html

Android Developers Blog (2017c). Introducing android 8.0 oreo. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2017/08/introducing-android-8-oreo.html

Android Developers Blog (2018). Introducing android 9 pie. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2018/08/introducing-android-9-pie.html

Android Developers Blog (2019a). Expanding target api level requirements in 2019. Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2019/02/expanding-target-api-level-requirements.html

Android Developers Blog (2019b). Welcoming android 10 ! Repéré le 8 novembre 2019 à https ://android-developers.googleblog.com/2019/09/welcoming-android-10.html

Android Developers Blog (2020). Turning it up to 11. Repéré le 29 octobre 2020 à https ://android-developers.googleblog.com/2020/09/android11-final-release.html

Android Open Source Project (n.d.). Codenames, tags, and build numbers. Repéré le 8 novembre 2019 à https ://source.android.com/setup/start/build-numbers

Briand, L., El Emam, K. & Morasca, S. (1996). On the application of measurement theory in software engineering. *Empirical Software Engineering*, *1*, 61–88. doi :10.1007/BF00125812

Cambridge Dictionary Online (2020). predictability. Repéré le 9 mars 2020 à https ://dictionary.cambridge.org/dictionary/english/predictability

Geiger, F.-X., Malavolta, I., Pascarella, L., Palomba, F., Di Nucci, D. & Bacchelli, A. (2018). A graph-based dataset of commit history of real-world android apps. Dans *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18 (pp. 30–33). New York, NY. doi :10.1145/3196398.3196460

GlobalStats (2019). Mobile operating system market share worldwide. Repéré le 14 octobre 2019 à https ://gs.statcounter.com/os-market-share/mobile/worldwide/2019

Gradle (n.d.a). Build script basics. Repéré le 2 juin 2020 à https ://docs.gradle.org/current/userguide/tutorial$_{u}sing_{t}asks.html$

Gradle (n.d.b). Writing build scripts. Repéré le 2 juin 2020 à https ://docs.gradle.org/current/userguide/writing$_{b}uild_{s}cripts.html$

Guilardi, D., Nicacio, J., Napoleao, B. & Petrillo, F. (2020a). Androidproptracker : Mining lifetime properties of android projects. Dans *IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems* (pp. 23–26). Seoul, Republic of Korea. doi :10.1145/3387905.3388606

Guilardi, D., Nicacio, J., Napoleao, B. & Petrillo, F. (2020b). Are apps ready for new android releases ? Dans *IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems* (pp. 66–76). Seoul, Republic of Korea. doi :10.1145/3387905.3388598

Han, D., Zhang, C., Fan, X., Hindle, A., Wong, K. & Stroulia, E. (2012). Understanding android fragmentation with topic analysis of vendor-specific bugs. Dans *2012 19th Working Conference on Reverse Engineering* (pp. 83–92). Kingston, ON. doi :10.1109/WCRE.2012.18

Harman, M., Jia, Y., Zhang, Y. & Bla, B. (2012). App store mining and analysis : Msr for app stores. Dans *IEEE/ACM 9th IEEE Working Conference on Mining Software Repositories (MSR)* (pp. 108–111). Zurich, Switzerland. doi :10.1109/MSR.2012.6224306

He, D., Li, L., Wang, L., Zheng, H., Li, G. & Xue, J. (2018). Understanding and detecting evolution-induced compatibility issues in android apps. Dans *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018 (pp. 167–177). New York, NY. doi :10.1145/3238147.3238185

Huang, H., Wei, L., Liu, Y. & Cheung, S.-C. (2018). Understanding and detecting callback compatibility issues for android applications. Dans *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018 (pp. 532–542). New York, NY. doi :10.1145/3238147.3238181

Joorabchi, M. E., Mesbah, A. & Kruchten, P. (2013). Real challenges in mobile app development. Dans *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 15–24). Baltimore, MD. doi :10.1109/ESEM.2013.9

Kosarevsky, S. & Latypov, V. (2015). *Mastering Android NDK*, (1er éd., pp. 21-22). Birmingham, United Kingdom : Packt Publishing.

Li, L., Gao, J., Bissyandé, T. F., Ma, L., Xia, X. & Klein, J. (2020). Cda : Characterising deprecated android apis. *Empirical Software Engineering*, pp. 2058–2098. doi :10.1007/s10664-019-09764-z

Merriam-Webster's Collegiate Dictionary Online (1999). readiness. Repéré le 3 janvier 2020 à https ://www.merriam-webster.com/dictionary/readiness

Montgomery, D. C., Peek, E. A. & Vining, G. G. (2012). *Introduction to Linear Regression Analysis*, (5e éd., pp. 9-12). Hoboken, NJ : John Wiley & Sons, Inc.

Mutchler, P., Safaei, Y., Doupé, A. & Mitchell, J. (2016). Target fragmentation in android apps. Dans *2016 IEEE Security and Privacy Workshops (SPW)* (pp. 204–213). San Jose, CA. doi :10.1109/SPW.2016.31

Oxford Learner's Dictionaries Online (2020). benchmark. Repéré le 14 février 2019 à https ://www.oxford-learners-dictionaries.com/definition/english/benchmark$_1$

Wei, L., Liu, Y. & Cheung, S.-C. (2016). Taming android fragmentation : Characterizing and detecting compatibility issues for android apps. Dans *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ASE 2016 (pp. 226–237). New York, NY. doi :10.1145/2970276.2970312

Wei, L., Liu, Y., Cheung, S. C., Huang, H., Lu, X. & Liu, X. (2018). Understanding and detecting fragmentation-induced compatibility issues for android apps. *IEEE Transactions on Software Engineering*, 1–1. doi :10.1109/tse.2018.2876439

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnel, B. & Wesslén, A. (2000). *Experimentation in Software Engineering*, (1er éd., pp. 10-41). Heidelberg, Germany : Springer science + business media.

Wu, L., Grace, M., Zhou, Y., Wu, C. & Jiang, X. (2013). The impact of vendor customizations on android security. Dans *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13 (pp. 623–634). New York, NY. doi :10.1145/2508859.2516728

Yang, G., Jones, J., Moninger, A. & Che, M. (2018). How do android operating system updates impact apps ? Dans *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*, MOBILESoft '18 (pp. 156–160). New York, NY. doi :10.1145/3197231.3197258

Yuan, T., Nagappan, M., Lo, D. & Hassan, A. E. (2015). What are the characteristics of high-rated apps ? a case study on free android applications. Dans *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Bremen, Germany. doi :10.1109/ICSM.2015.7332476

Zhou, X., Lee, Y., Zhang, N., Naveed, M. & Wang, X. (2014). The peril of fragmentation : Security

hazards in android device driver customizations. Dans *2014 IEEE Symposium on Security and Privacy* (pp. 409–423). San Jose, CA. doi :10.1109/SP.2014.33